AreaList Pro™

User Manual

FORESIGHT TECHNOLOGY

AreaList™ Pro

User Manual

Published World-Wide by Foresight Technology, Inc.

Foresight Technology, Inc.
4100 International Plaza
Suite 538
Fort Worth, TX 76109

AreaList Pro

Written by: Steve Antonakes
Manual: Rich Gay and Steve Antonakes

Software License and Limited Warranty

PLEASE READ THIS LICENSE CAREFULLY BEFORE USING THE SOFTWARE CONTAINED ON THE DISKS. BY USING THE SOFTWARE, YOU AGREE TO BECOME BOUND BY THE TERMS OF THIS AGREEMENT, WHICH INCLUDES THE SOFTWARE LICENSE AND WARRANTY DISCLAIMER (collectively referred to herein as the "Agreement"). THIS AGREEMENT CONSTITUTES THE COMPLETE AGREEMENT BETWEEN YOU AND FORESIGHT TECHNOLOGY, INC. IF YOU DO NOT AGREE TO THE TERMS OF THIS AGREEMENT, DO NOT USE THE SOFTWARE AND PROMPTLY RETURN THE PACKAGE FOR A FULL REFUND.

1. Ownership of Software. The enclosed manual and computer programs ("Software") were developed and are copyrighted by Foresight Technology, Inc. ("Foresight") and are licensed, not sold, to you by Foresight for use under the following terms, and Foresight reserves any rights not expressly granted to you. You own the disk(s) on which any software is recorded, but Foresight retains ownership of all copies of the Software itself. Neither the manual nor the Software may be copied in whole or in part except as explicitly stated below.

2. License. Foresight, as Licensor, grants to you, the LICENSEE, a non-exclusive, non-transferable right to use this Software subject to the terms of the license as described below:

   a. You may make backup copies of the Software for your use provided they bear the Foresight copyright notice.

   b. You may use this Software in an unlimited number of custom or commercial databases or applications created by the original licensee. No additional product license or royalty is required.

3. Restrictions. You may not distribute copies of the Software to others (except as an integral part of a database or application within the terms of this License) or electronically transfer the Software from one computer to another over a network. You may not distribute copies of the Software as an integral part of a development shell or non-compiled commercial data base. The Software contains trade secrets and to protect them you may not decompile, reverse engineer, disassemble, or otherwise reduce the Software to a human perceivable form. YOU MAY NOT MODIFY, ADAPT, TRANSLATE, RENT, LEASE, LOAN OR RESELL FOR PROFIT THE SOFTWARE OR ANY PART THEREOF.

4. Termination. This license is effective until terminated. This license will terminate immediately without notice from Foresight if

you fail to comply with any of its provisions. Upon termination you must destroy the Software and all copies thereof, and you may terminate this license at any time by doing so.

5. Update Policy. Foresight may create, from time to time, updated versions of the Software. At its option, Foresight will make such updates available to the Licensee.

6. Warranty Disclaimer. THE SOFTWARE IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED , INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. FORESIGHT DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIALS IN THE TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, CURRENTNESS OR OTHERWISE. THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE SOFTWARE IS ASSUMED BY YOU. IF THE SOFTWARE OR WRITTEN MATERIALS ARE DEFECTIVE YOU, AND NOT FORESIGHT OR IT’S DEALERS, DISTRIBUTORS, AGENTS, OR EMPLOYEES, ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION. However, Foresight warrants to the original Licensee that the disk(s) on which the Software is recorded is free from defects in materials and workmanship under normal use and service for a period of thirty (30) days from the date of delivery as evidenced by a copy of the receipt.

THIS IS THE ONLY WARRANT OF ANY KIND, EITHER EXPRESS OR IMPLIED, THAT IS MADE BY FORESIGHT ON THIS SOFTWARE PRODUCT. NO ORAL OR WRITTEN INFORMATION OR ADVICE GIVEN BY FORESIGHT, IT’S DEALERS, DISTRIBUTORS, AGENTS, OR EMPLOYEES SHALL CREATE A WARRANTY OR IN ANY WAY INCREASE THE SCOPE OF THIS WARRANTY, AND YOU MAY NOT RELY ON SUCH INFORMATION OR ADVICE. THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS. YOU MAY HAVE OTHER RIGHTS, WHICH VARY FROM STATE TO STATE.

7. Governing Law. This agreement shall be governed by the laws of the State of Texas.

Table of Contents

Table of Contents

Using the AreaList Pro Manual

General information about the AreaList Pro user interface is discussed in “The AreaList Pro User Interface” on page 11.

An overview of the AreaList Pro commands and useage is covered in “Developing with AreaList Pro” on page 21.

Commands are organized by topic into individual chapters. Each chapter begins with an overview of the topic, and how to use the different commands. Each command is then covered in detail, and examples provided.

If you are having troubles with AreaList Pro, review the chapter “Troubleshooting” on page 197. If you are unable to resolve a problem using this manual, you can contact our Technical Support Department. See “Technical Support” on page 203.

Cross-Referencing Format

Each time a command is used, a cross-reference is given in parenthesis to let you quickly find the definition for the command.

Command Lists

There are two lists of commands at the back of the manual. One is organized by chapter, and the other alphabetical. These lists include the parameters for each command, and the page number for the command definition.

Command Descriptions and Syntax

Each AreaList Pro command has a syntax, or rules, that describe how to use the command in your 4D database. For each command, the name of the command is followed by the command’s parameters. The parameters are enclosed in parenthesis, and separated by semicolons. Following the command syntax description, an explanation of the command’s parameters is provided. For each parameter, the type of the parameter and a description is shown. Several examples are provided for each of the commands, showing examples of the syntax as well as how the various commands are used together.

The first parameter for each command is the name of the AreaList Pro object on the layout. This parameter is a long integer, and is required to allow the commands to operate on the correct object.

Some parameters are used by AreaList Pro to return a value. Do not use local variables for these parameters, because 4D doesn’t support returning values from an external into a local variable. Use a global variable in 4D v2, and in 4D v3 or later, a process or inter-process variable.

About AreaList Pro

AreaList Pro is an easy-to-use tool for implementing scrolling lists on 4th Dimension layouts. Because AreaList Pro is an external, it is very fast, and provides capabilities not available to the developer using native 4D commands and objects, such as horizontal scrolling, user-resizable columns, automatic column sizing and formatting, copy to the clipboard, drag and drop interfaces, and more. The array contents can even be altered directly by entering data into the AreaList Pro area using typed characters and popup menus, with full control over data entry. Operation is extremely fast, and control objects (scroll bars, buttons, etc.) follow the Macintosh interface.

Data is passed to AreaList Pro using 4D arrays, or field numbers. If only two columns need to be displayed, create two arrays or specify two fields and pass them as parameters to AreaList Pro. No string parsing or other contortions are needed.

AreaList Pro can be used with just one command — no special formatting is required. For those cases when formatting is needed, several optional commands give you complete control over the appearance of the area.

Special tools are implemented for the developer who desires to customize the appearance and configuration of AreaList Pro, allowing the customization to be implemented rapidly.

AreaList Pro v6.0 requires 4th Dimension v2.2.3 or higher, and is compatible with databases complied with the 4D Compiler. AreaList Pro v6.0 is fully compatible with 4th Dimension v3.5 and 4D Server v1.5. The field display capability available with version 6.0 requires 4D version 3.5.3, or 4D Server 1.5.3 or later.

AreaList Pro v6.0 also requires System 7 on a Macintosh and Windows 3.1, Windows 95 or Windows NT on a PC.

About AreaList Pro

Installing and De-Installing AreaList™ Pro

AreaList Pro must be installed (and de-installed) using the AreaList Pro Installer or the Mac4DX or Win4DX installation method described herein. Do not use the 4D External Mover or ResEdit to install or remove AreaList Pro.

Installing AreaList Pro for Macintosh

The AreaList Pro for Macintosh disk contains two different versions of the external. AreaList™ Pro v6.0 Installer will install a version that will run native on a 680x0 processor based Macintosh. The file AreaList Pro v6.0 contains a version of AreaList Pro that can run native on both a Power Macintosh and a 680x0 Macintosh. This second version can only be used with 4th Dimension v3.2.5 or later.

These two versions are installed differently. The installation method used has no effect on the way that AreaList Pro works. When deciding where to install AreaList Pro, you should consider the convenience of that location, and be sure to avoid installing it in more than one of: the Proc.Ext file or the 4D structure file.

One final caution with the installation of AreaList Pro and other exter-nals… if you are installing in a Proc.Ext or the Structure file, be sure to keep all of the externals for a database in one place only. Mixing some of the database externals in the Proc.Ext file and some in the struc-ture file can cause your database to crash due to resource ID conflicts.

For more information about 4th Dimension externals and their installation, please refer to the 4th Dimension manuals.

Installing AreaList Pro for Macintosh Using the Installer

This installation method is required when using versions of 4D prior to 3.2.5. The AreaList™ Pro v6.0 Installer will allow you to install AreaList Pro directly into the database structure file or into a Proc.Ext file. A Proc.Ext file is recognized by 4th Dimension as if it were part of a database, and can contain one or more externals, making them available to a database in the same folder as the Proc.Ext file.

This section discusses installation of AreaList Pro into a new Proc.ext file, or an existing Proc.ext or structure file.

To Install AreaList Pro into a New Proc.Ext File

1 Open the AreaList Pro Installer. The installation window is displayed.
2 Click the New button on the left. The Installer displays a standard Macintosh file dialog box with Proc.Ext already entered in the filename area.
This dialog box allows you to specify the location to save the Proc.Ext file on your hard disk. Normally, you will keep it in the same folder as your 4th Dimension application.
3 Click Save.
The Installer creates the Proc.Ext file and saves it in the location you specified.

The message area on the Installation dialog shows that the Proc.Ext file you just created is now selected, but that AreaList Pro is not yet installed.
4 Click Install.
The AreaList Pro Installer installs AreaList Pro into the Proc.Ext file. This process takes no more than a few seconds.
5 When the message area shows that AreaList Pro is installed, click Quit. You are returned to the Finder.

To Add AreaList Pro to a Proc.Ext File or a 4D Structure File

If you have already created a Proc.Ext file, or would prefer to install AreaList Pro directly into a 4D structure file, you can use the installer to add AreaList Pro to it.

1 Open the AreaList Pro Installer. The installation window is displayed.
The message area indicates that there is no file currently selected. In a moment, this area will identify the file in which you will be install-ing AreaList Pro.
2 Click Open.
The Installer displays the standard Macintosh file dialog box. The list displays Proc.Ext files, 4D structure files, and folders.

Note that in System 7, you can open a structure or Proc.Ext file directly from the Finder by dragging the file onto the AreaList Pro Installer.
3 Select the Proc.Ext File or 4D Structure File in which you want to install AreaList Pro.
4 Click Open.
The AreaList Pro Installer opens the selected file.

The message area on the AreaList Pro Installer window now shows the name of the file you selected, and indicates whether or not AreaList Pro is currently installed in this file. If AreaList Pro is installed, the version will also be displayed.

5 Click Install.
AreaList Pro is installed into the selected file. This process takes only a few seconds.

6 When the message area shows that AreaList Pro is installed, click Quit.
You are returned to the Finder.

Installing the Mac4DX Version of AreaList Pro

If you are using 4th Dimension v3.2.5 or higher, a new method for installing externals exists in addition to the above described method. You can now create a folder named Mac4DX in the same folder as the structure file, and simply drop special external files into that folder. Use of this method avoids some external conflicts, and it is the method used to install the AreaList Pro Power Mac version.

In addition to the AreaList Pro installer described in the previous sec-tions, your AreaList Pro disk also contains a file entitled AreaList Pro™ v6.0. This is a FAT version of AreaList Pro, which means that it will run native on either a Power Macintosh or a 680x0 Macintosh. This version can only be used with 4th Dimension v3.2.5 or later.

To install the Power Macintosh version of AreaList Pro:

1 Create a folder inside the same folder as your 4D Structure file.

2 Make the name of this folder Mac4DX. (If this folder already exists, you can skip steps 1 and 2.)

3 Copy the file AreaList™ Pro v6.0 into this folder.

Installing the Windows Version of AreaList Pro

Your AreaList Pro for Windows installer will create two files containing the external: AreaList.4DX and AreaList.RSR. The former contains the executable code for the external, and the latter contains the external's resources. They are located in the installed directory \AreaList Pro\WIN4DX.

To install the Windows version of AreaList Pro:

1 Execute the file A:\SETUP.EXE on the AreaList Pro for Windows disk.
This will install a AreaList Pro directory structure on your disk.

2 Create a directory inside the same directory as your 4D Structure file.

3 Make the name of this directory WIN4DX (if this directory already exists, you can skip steps 2 and 3.)

4 Copy the files AreaList.4DX, AreaList.RSR, and W4D_Drag.DLL from the installed directory \AreaList Pro\WIN4DX\ into the directory you created.
W4D_Drag.DLL is needed for dragging in AreaList Pro on Windows. This needs to be placed in the Win4DX folder along with the AreaList.4DX and AreaList.RSR files. If you do not place the W4D_Drag.DLL in the Win4DX folder you will not be allowed to drag with AreaList Pro. If you do not plan to do any dragging with AreaList Pro you do not need this file in your Win4DX folder.

De-Installing AreaList Pro from Macintosh Versions of 4D

The AreaList Pro Installer allows you to de-install AreaList Pro from a Proc.Ext file or 4D Structure file.

To De-Install AreaList Pro from a Macintosh Proc.ext or Structure File

1 Open the AreaList Pro Installer.
The installation window is displayed.

The message area indicates that there is no file currently selected. In a moment, this area will identify the file in which you will be de-installing AreaList Pro.

2 Click Open.
The Installer displays the standard Macintosh file dialog box. The list displays Proc.Ext files, 4D structure files, and folders.

3 Select the Proc.Ext File or 4D Structure File from which you want to de-install AreaList Pro.

4 Click Open.
The selected file is opened.

The message area on the AreaList Pro Installer window now shows the name of the file you selected, and indicates whether or not AreaList Pro is currently installed in this file. If AreaList Pro is already installed in the file, the Remove button is enabled.

5 Click Remove.
The Installer de-installs AreaList Pro from the selected file. This pro-cess takes only a few seconds.

6 When the message area shows that AreaList Pro is not installed, click Quit.
You are returned to the Finder.

De-Installing the Mac4DX version of AreaList Pro

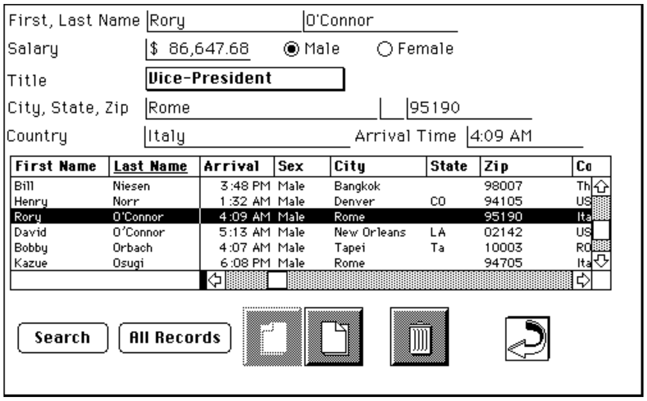
To de-install the Power Mac version of AreaList Pro, simply remove the file AreaList™ Pro v6.0 from the Mac4DX folder.

De-Installing the Windows version of AreaList Pro

To de-install the Windows version of AreaList Pro, simply remove the files AreaList.4DX and AreaList.RSR from the WIN4DX directory that is at the same level as your structure file.

The AreaList Pro User Interface

AreaList Pro displays a scrolling area on 4D layouts, as shown below.



First, Last Name Rory O'Connor

Salary $ 86,647.68 ● Male ○ Female

Title Vice-President

City, State, Zip Rome 95190

Country Italy Arrival Time 4:09 AM

First Name	Last Name	Arrival	Sex	City	State	Zip	Co
Bill	Niesen	3:48 PM	Male	Bangkok		98007	Th
Henry	Norr	1:32 AM	Male	Denver	CO	94105	US
Rory	O'Connor	4:09 AM	Male	Rome		95190	Ita
David	O'Connor	5:13 AM	Male	New Orleans	LA	02142	US
Bobby	Orbach	4:07 AM	Male	Tapei	Ta	10003	RO
Kazue	Osugi	6:08 PM	Male	Rome		94705	Ita

Headers

Above the scrollable AreaList Pro area, there may be a row of cells called the Header area. This area is usually used to contain a description of the data displayed in each of the columns. The Header area is also used to control the sorting of the data and column dragging, if these features are enabled.The Header area is not editable by the user, and will not scroll vertically with the rest of the AreaList Pro area.

The user can click on a header to sort the list using that column. See “Sorting” on page 13.

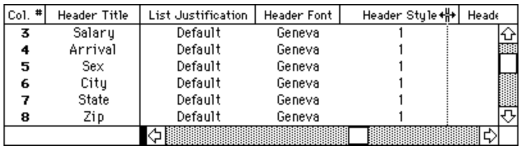
The user can click and drag a header to move a column to a new location. See “Drag and Drop” on page 15.

Footers

Below the scrollable AreaList Pro area, there may be a row of cells called the Footer area. This area can be used to store information about the column, such as the total of a numeric column’s data. The Footer area is not editable by the user, and will not scroll vertically with the rest of the AreaList Pro area.

Column Widths

The user can resize any column by moving the arrow over the line dividing the columns in the header area. The pointer will change to , the shape 4D uses in the Quick Report Editor for column resizing. Drag this column divider to resize the column.



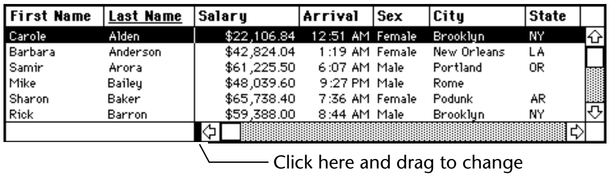
Col. #	Header Title	List Justification	Header Font	Header Style	Heade
3	Salary	Default	Geneva	1	
4	Arrival	Default	Geneva	1	
5	Sex	Default	Geneva	1	
6	City	Default	Geneva	1	
7	State	Default	Geneva	1	
8	Zip	Default	Geneva	1	

AL_SetColOpts (page 62) can be used to disable this feature.

Note: A column can not be resized to greater than the width of the list area minus 20 pixels.

Column Locking

One or more columns on the left side of the list can be locked in place to prevent them from scrolling horizontally. The user can adjust the lock position by dragging the Column Lock control, shown below.



First Name	Last Name	Salary	Arrival	Sex	City	State
Carole	Alden	$22,106.84	12:51 AM	Female	Brooklyn	NY
Barbara	Anderson	$42,824.04	1:19 AM	Female	New Orleans	LA
Samir	Arora	$61,225.50	6:07 AM	Male	Portland	OR
Mike	Bailey	$48,039.60	9:27 PM	Male	Rome	
Sharon	Baker	$65,738.40	7:36 AM	Female	Podunk	AR
Rick	Barron	$59,388.00	8:44 AM	Male	Brooklyn	NY

Click here and drag to change the column lock position

When columns are locked and the user clicks in the horizontal scroll bar, the locked columns will not scroll. This capability is similar to the Freeze Panes feature in Excel. When the column lock position is adjusted, the list will automatically scroll to the full left position to provide feedback to the user.

Rows with Multiple Lines of Text

AreaList Pro allows individual rows in the list area to contain more than one line of text; however, all rows in the area will be of the same height.

Color

AreaList Pro allows the entire range of 256 colors in the 4D palette, or the 10 colors of the built-in AreaList Pro palette. Foreground colors can be applied to columns, individual rows, cells, headers, and footers. Background colors can be applied to the list area, individual rows, cells, the header area, and the footer area.

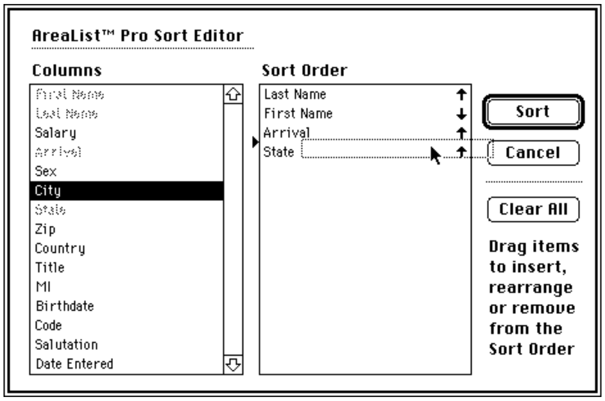
Styles

AreaList Pro supports all standard styles used by the Macintosh, including **Bold**, *Italic*, Underline, Outline, Shadow, Condensed, or extended, or any combination of these. These styles may be applied to columns, headers, footers, individual rows, or cells in an AreaList Pro area.

Sorting

The list can be sorted in ascending (A to Z) order by clicking a column header, and sorted in descending order by Option-clicking the column header. Sorting the list actually sorts the 4D columns displayed in the list. An underlined header indicates the current sort column. If a column contains a picture, clicking its column header will cause it to highlight, but no sorting will occur.

In addition to clicking a column header to sort, there is a sort editor available to allow sorting on multiple columns (such as Last Name, First Name). To access this feature, Command-click the header area of the AreaList Pro object. The dialog box shown below is displayed.



AreaList™ Pro Sort Editor

Columns

First Name
Last Name
Salary
Arrival
Sex
City
State
Zip
Country
Title
MI
Birthdate
Code
Salutation
Date Entered

Sort Order

Last Name ↑
First Name ↓
Arrival ↑
State ↑

Drag items to insert, rearrange or remove from the Sort Order

The area on the left is a list of the columns displayed in the AreaList Pro object. An item can be added to the Sort Order list on the right by dragging it over the rectangle on the right or by double-clicking. To remove an item from the Sort Order list, drag it outside of the Sort Order list area.

To change the direction of the sort

1 Click the arrows to the right of each item in the Sort Order list (up arrow is ascending order, down arrow is descending.) Although picture columns cannot be sorted, they will appear in the list of columns. However, the item(s) for the picture column(s) will be disabled and cannot be dragged into the Sort Order list.

2 Click the Sort button.

When displaying fields, the following features are present.

- Indexed fields will be bold in the Sort Editor.
- Fields from related one files will be dimmed in the Sort Editor.

Scrolling

The list can be scrolled in the following ways:

- Clicking the arrows and other scroll controls.
- Using the keyboard arrow keys. Each press of the arrow key will scroll the list one row or column in the direction corresponding to that key. Option-Arrow will scroll the list to the top, bottom, far left, or far right.
- By typing on the keyboard. As characters are typed, the current sort column will be used to vertically scroll the list. If there is a pause between typed characters, then the scrolling action will “reset.” The pause time is equal to the double-click time set in the Macintosh Control Panel. *This feature is disabled when displaying fields. See “Specifying the Fields to Display” on page 27 for more information.*
- Clicking the list area and dragging the mouse arrow outside of the list area. This action will scroll both horizontally and vertically.
- Dragging a row or column. When dragging a row or column within an AreaList Pro object, or to another valid AreaList Pro object, the destination area may scroll. See “Drag and Drop” on page 15.

Selection

The user can create a selection in an AreaList Pro area in one of several ways: single line, multiple line, single cell, and multiple cell. In single line selection, clicking a line will select that line, and only one line can be selected at a time. In multiline mode, the user can select multiple lines by dragging, Shift-clicking (continuous selection), or Command-clicking (discontinuous selection). In single cell mode, clicking a cell will select only that cell, not the entire row. In multiple cell mode, the user can select none, one, or many cells. The effect of any of these methods on already selected rows or cells will be the same: the rows or cells will be deselected.

The Edit menu Select All command will select all lines when the multiple line selection option has been enabled, or select all cells when the multiple cell selection option has been enabled.

Copy to Clipboard

Lines selected in an AreaList Pro object can be copied to the Clipboard via the Edit menu Copy command. Because of the limitations of the Macintosh clipboard — when a selection of lines are copied to the clipboard, pictures will not be copied; a blank field will appear on the clipboard where the picture would have been.

Copying rows to the clipboard will not be allowed when displaying fields. The Copy menu item will be disabled when fields are displayed.

Drag and Drop

The Drag and Drop feature of AreaList Pro allows the User to drag a row or column in an AreaList Pro object to a different position within that same area. This feature may also be used to drag a row or column to a different AreaList Pro object, to a CalendarSet object, or to an ***%AL_DropArea*** (see “DropArea” on page 155), on the same layout or a different layout.

To Drag a Row

AreaList Pro allows row dragging to be initiated by either Option-clicking on a row and dragging it, or by just dragging the row, depending on how AreaList Pro is configured. When the row is clicked on and dragged the row will move freely with the pointer. If the row is not accepted by the destination object a rectangle will zoom back to the origin of the drag.

The user selects multiple rows by command-clicking or shift-clicking. If the *DragRowWithOptKey* option of ***AL_SetDrgOpts*** (page 137) is set to 1, then the user can also select multiple rows by dragging. Once the row(s) are selected, the user may click (or option-click) to drag them. An outline of the row(s) will follow the pointer (cursor) location until the mouse is released.

To Drag a Column

AreaList Pro allows column dragging to be initiated by clicking the column header and dragging. If the *UserSort* option of ***AL_SetSortOpts*** (page 69) is disabled, column dragging will begin immediately, and an outline of the column will appear. If user sorting is enabled, the drag begins when the mouse pointer is greater than 20 pixels to the left or right of the column, or greater than 30 pixels above or below the column header. When the column is clicked on and dragged the column will move freely with the pointer. If the column is not accepted by the destination object a rectangle will zoom back to the origin of the drag.

Dragging to a Row

The list will scroll when the arrow is moved within the number of pixels of the AreaList Pro object specified in ***AL_SetDrgOpts*** (page 137). A small triangle will appear adjacent to the left side of the destination object, indicating the insertion position.

Dragging to a Column

The list will scroll when the arrow is moved within the number of pixels of the AreaList Pro object specified in ***AL_SetDrgOpts*** (page 137). A small triangle will appear adjacent to the top of the destination object, indicating the insertion position.

To Drag a Cell

The user drags a cell by clicking upon it and dragging it. An outline of the cell will follow the pointer (cursor) location until the mouse is released.

Dragging to a Cell

When enabled, the user can drop an item as a row, as a column or as a cell. If the destination is a cell, an outline will be shown inside the cell that the cursor is over to indicate where the item will be dropped.

See “Dragging Commands” on page 129 for more information.

Enterability

Initiating Data Entry

Data entry using typed characters may be initiated on an AreaList Pro object by several programmable methods, all of which require clicking in the cell with or without a modifier key. For example, data entry on a given cell could be initiated upon a single click in that cell, a double click, or a double click along with the Option, Command, Shift, or Control key.

Entering Data

Once typed data entry is initiated, standard editing functions can be performed on the selected cell, including the Edit menu commands Cut, Copy, Paste, Clear, Select All, and Undo. This is true for cells containing pictures, also (except Select All.) Alphanumeric data being edited will always appear left-justified, regardless of the column's display justification. The I-Beam pointer can be dragged across the data in the cell to select a portion or all of the data.

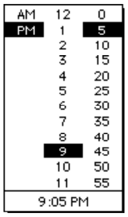
If string data is entered, the system beep will sound for every character typed past the maximum string length, and the typed character will be ignored. (Note: there are special programming considerations concerning this feature. See "Maximum Length of a String Exceeded" on page 106.) If a string which exceeds the maximum string length is pasted into a cell, it will appear in the cell in its entirety, but will be truncated to the maximum string length when the insertion point leaves that cell.

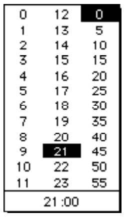
Boolean data is represented during data entry by either radio buttons or a checkbox. This data may be entered via several methods, including using the space bar, using the key combinations t/f, T/F, y/n, Y/N, or the first letters, upper and lower case, of values specified in the format for the boolean data entry column. When entering other types of data, as in 4th Dimension, data entry may be restricted to specific requirements via the use of filters.

Data Entry Using Popups

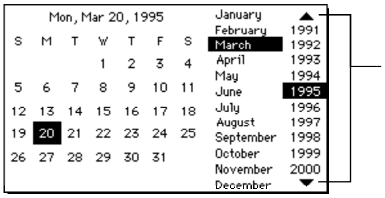
AreaList Pro also has the ability to perform data entry using popup menus for column data types other than picture or boolean. Popup menus will appear as small buttons on the right side of the cell which will be labeled with a downward pointing triangle. The items contained in the popup menu represent the possible values for that cell, which are determined by you. However, for time or date information, a special popup menu will allow the user to choose appropriate values for these data types. The presence of a popup menu in a cell does not necessarily prohibit the ability to enter typed characters.

The time menu is shown below. To select a time, the user should begin on the left side of the popup, first selecting AM or PM, then the hour, then the minutes. This menu will appear slightly different depending on your system settings for the time format (using a 24 hour clock, for example), but the method of selecting the time will remain basically the same.





The date popup menu selects a date using a slightly different method: the user begins on the right side of the popup, selects the year, then month, and last, the day.



Click here to scroll the years displayed on the popup

Moving the Current Entry Cell

AreaList Pro speeds data entry by making it easy to move to other enterable cells once data entry is initiated. Since enterability is determined on a column by column basis, the cells adjacent to the current data entry cell may not be enterable. AreaList Pro handles this situation by using the Tab key to move to the next enterable cell to the right. A Shift-Tab combination will move data entry to the next enterable cell to the left. If there isn’t an enterable cell on the same row, these key combinations will move the data entry cursor to the next or previous row, respectively.

The Return key can be used in two ways during data entry. Normally, when the Return key or Shift-Return key is used, data entry will be moved to the next or previous row in the same column as the current data entry cell. However, in some cases the Return key may be used to enter a carriage return character into a text cell. As a default, the Return key moves the data entry position. You may choose to configure the Enter key to function the same as either the Return key or the Tab key, and also have the option of causing the Arrow keys to move the insertion point from cell to cell.

Exiting Data Entry

The user may exit data entry mode by using the mouse to click on another layout object, an AreaList Pro control or header, or a non-enterable column in the AreaList Pro area. However, if the data that was entered was invalid, the cell cannot be exited until valid data is entered. This is determined by the entry finished callback procedure. See “Using Callback Procedures During Data Entry” on page 108.

Enterability for Fields

Columns containing fields from a related one file will not be enterable either by typing or by using popups.

Resizable Windows with AreaList Pro

You can configure an AreaList Pro object to be resizable on a resizable window. When placed in the lower right portion of a window, AreaList Pro will draw a size box in the lower right hand corner of the window. Click on this box and drag to resize the AreaList Pro object and its window.

Developing with AreaList Pro

Creating an AreaList Pro object on a Layout

Implementing AreaList Pro in your 4D databases is very easy; in fact, displaying data in a AreaList Pro area can be accomplished with only one external command. The AreaList Pro object is drawn on a 4D layout using the variable object tool. 4D opens the Definition dialog for the object, which is where the object is named and configured. The name will be used as a parameter for the AreaList Pro commands.

Be careful to never have two AreaList Pro objects with the same name on a 4D layout.

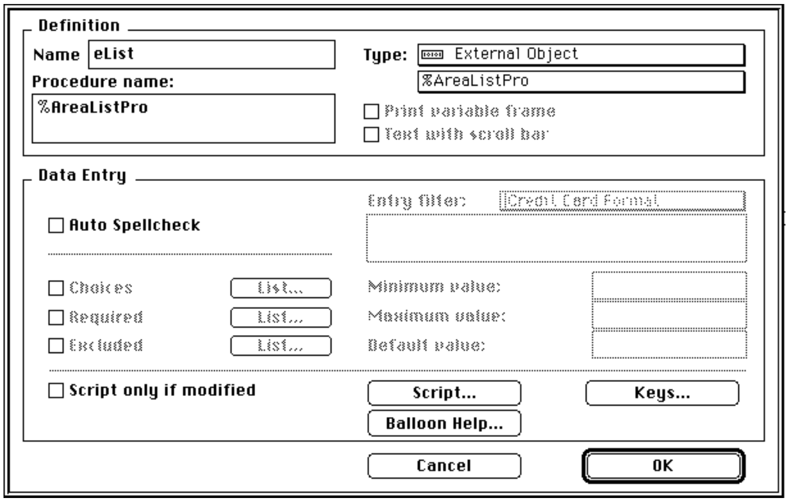
To configure a variable object as an AreaList Pro object

1 Create a variable object on a layout.
4D displays the variable object configuration dialog.

2 Select the External Object type.
The popup next to the Object type popup will include the %AreaListPro external.

3 Choose the “%AreaListPro” item from the popup next to the Type popup.
4D will enter it as the Procedure name on the right side of the dialog, as shown below (the dialog shown below is the one 4D v3 uses for smaller screen sizes; the actual dialog presented may be different depending on the version of 4D you are using, as well as your Macintosh configuration).

4 Name the variable.
This name will be used as the first parameter to many of the AreaList Pro commands.
Note: This variable must be a process variable, not an interprocess variable (i.e., the name cannot begin with the “◊” character).



Definition

Name: eList

Type: External Object

Procedure name: %AreaListPro

%AreaListPro

□ Print variable frame

□ Text with scroll bar

Data Entry

Entry filter: Credit Card Format

□ Auto Spellcheck

□ Choices

□ Required

□ Excluded

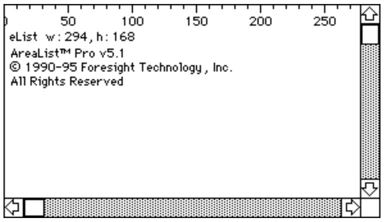
Minimum value:

Maximum value:

Default value:

□ Script only if modified

5 Click the OK button.
The AreaList Pro object is drawn in the layout editor.



eList w: 294, h: 168
AreaList™ Pro v5.1
© 1990-95 Foresight Technology, Inc.
All Rights Reserved

The first line of text contains the name of the object and its pixel dimensions, and the remaining lines are the Copyright notice. If the object is small, the horizontal and vertical scroll bars are not displayed in the Layout Editor, but everything will function correctly. The display of the object name, pixel dimensions, Copyright notice, and scroll bars is an indication that the object has been properly created and named.

AreaList Pro Object Dimensions

AreaList Pro provides information to allow you to properly size the AreaList Pro area and to align it with other objects on the layout in the 4th Dimension Design environment. A scale at the top of the external object indicates the pixel width of the AreaList Pro object. This may be used to align other layout objects which appear adjacent to the AreaList Pro object. Displayed next to the object's name is the width and the height of the object as it is drawn on the layout. These values include the entire area displayed by AreaList Pro, including the header and scroll bars, and they will be updated whenever the object is resized.

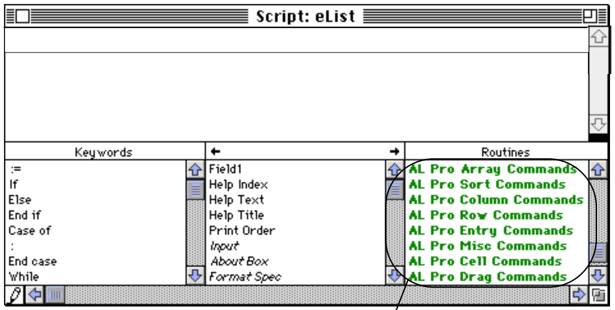
See “AreaList Pro Height” on page 30 for additional information about controlling the height of an AreaList Pro object.

Creating an %AL_DropArea on a Layout

To create AreaList Pro's ***%AL_DropArea*** (page 155) external area, follow the same method as is used to create an ***%AreaListPro*** area, only select ***%AL_DropArea*** from 4D's external area popup. No text other than the area name will appear inside the ***%AL_DropArea*** object.

Using the AreaList Pro Commands

The AreaList Pro Commands are used in the same way that a 4D command is used. Parameters are separated by the semicolon character (";"). You can access the AreaList Pro commands in the Procedure editor list. Near the bottom of the list, below the area which contains the global procedures, there are seven AreaList Pro command topics as shown below.



Click on a topic to display a popup menu listing commands for that topic

Clicking on a topic presents a popup menu of the AreaList Pro commands available. Simply select a command, and 4D will enter it for you at the current cursor position.

You can also type the command directly into the procedure.

Command Descriptions and Syntax

Each AreaList Pro command has a syntax, or rules, that describe how to use the command in your 4D database. For each command, the name of the command is followed by the command's parameters. The parameters are enclosed in parenthesis, and separated by semicolons. Following the command syntax description, an explanation of the command's parameters is provided. For each parameter, the type of the parameter and a description is shown. Several examples are provided for each of the commands, showing examples of the syntax as well as how the various commands are used together.

The first parameter for each command is the name of the AreaList Pro object on the layout. This parameter is a long integer, and is required to allow the commands to operate on the correct object.

Causing an AreaList Pro Object's Script to Execute

4D doesn't provide a "selectable" external area with a way for the external area to cause it's attached script to run. Since you will almost always want to have an AreaList Pro's object script run when the user takes an action on the area, such as clicking to select a line, a mechanism is provided to accomplish this. See "AreaList Pro's PostKey" on page 145.

Developer Alert

If the first parameter passed to any AreaList Pro command is not the object reference, an alert box will appear, informing you of the syntax error. If this object reference is a PrintList Pro area or another, external area, AreaList Pro will also pass this information to you.

Return Parameters from AreaList Pro Procedures

Some parameters are used by AreaList Pro to return a value. Do not use local variables for these parameters, because 4D doesn't support returning values from an external into a local variable. Use a global variable in 4D v2, and in 4D v3, a process or interprocess variable.

Using Pointers with AreaList Pro Commands

When writing generic AreaList Pro code, dereferenced pointers can be used for the parameters in AreaList Pro commands. However, this is not needed for any of the parameters except for the arrays passed using the obsolete commands ***AL_SetArrays*** (page 179) and ***AL_InsertArrays*** (page 181). *Please read the section "Setting Arrays" on page 177 for more information.*

Configuration Commands

AreaList Pro lets you display arrays or fields. This chapter discusses commands used to display arrays, as well as general configuration commands. See “Field and Record Commands” on page 95 for a complete discussion on displaying fields in an AreaList Pro object.

Specifying the Arrays to Display

4D arrays are passed directly to AreaList Pro for display via the ***AL_SetArraysNam*** (page 41) and ***AL_InsArrayNam*** (page 43) commands. These should be performed in the **Before** or **During** phase of layout execution, depending upon the desired appearance of the AreaList Pro area upon initial display of the layout. You do not have to setup or configure AreaList Pro in the **Before** phase of a layout; this can all be accomplished in the **During** phase. If no AreaList Pro setup is performed in the **Before** phase, nothing will be displayed in the space occupied by the external area until setup occurs in the **During** phase.

Whether the AreaList Pro columns are set in the **Before** phase or in the **During** phase, the setup of an AreaList Pro area must follow one main rule:

AL_SetArraysNam *or* ***AL_InsArrayNam*** *must be called before any other AreaList Pro commands are executed.*

This is necessary to provide AreaList Pro with an opportunity to allocate the data structures necessary to store formatting information for each column. These data structures are allocated on a per column basis, and ***AL_SetArraysNam*** for a given column (or ***AL_InsArrayNam***) must be executed before the appearance, enterability, style, or any other property of that column can be specified. If the ***AL_SetArraysNam*** or ***AL_InsArrayNam*** command is incorrectly used, an error code indicating the problem will be returned:

Valu e	Error Code	Action
0	No error	n/a
1	Not an array	check to make sure all arrays are correctly typed

Value	Error Code	Action
2	Wrong type of array	pointer and two-dimensional arrays are not allowed
3	Wrong number of rows	make sure that all arrays have the same number of elements
4	Maximum number of arrays exceeded	100 arrays is the maximum
5	Not enough memory	Increase 4D's RAM partition, or change your approach to use fewer or smaller arrays

Up to 255 arrays can be displayed by AreaList Pro, with up to fifteen columns specified in each call to ***AL_SetArraysNam*** or ***AL_InsArrayNam***. The position of the first array, *ColumnNum* , and the number of arrays, *NumArrays*, are also specified in these commands. *All array types except for pointer and two dimensional arrays, are allowed, and all arrays must have the same number of elements.*

Note: The maximum number of rows is 32,750. A future version of AreaList Pro will support the display of up to 8,000,000 rows.

In addition to standard single-dimension arrays, one dimension of a two-dimensional array may be passed to ***AL_SetArraysNam*** (page 41) or ***AL_InsArrayNam*** (page 43). For example: “My2DArray{1}” may be passed as Array1.

While similar in purpose, the commands ***AL_SetArraysNam*** and ***AL_InsArrayNam*** affect previously specified arrays in different ways. In the second or any subsequent executions of ***AL_SetArraysNam***, if *ColumnNum* is the number of a currently existing column, then it and any subsequent columns will be replaced by the arrays specified in the command. However, ***AL_InsArrayNam*** will actually insert the new arrays specified, and simply move existing arrays over to accomodate them. In both commands, the column number specified must either already exist or be the next higher column number available; no column numbers can be skipped.

For more information about adding, replacing and deleting arrays, read “Inserting and Deleting Arrays”, below.

Inserting and Deleting Arrays

After the initial setup and display of the AreaList Pro area, you may want to insert, remove, or replace arrays in the currently displayed AreaList Pro object. To accomplish this, AreaList Pro provides the commands ***AL_InsArrayNam*** (page 43), ***AL_RemoveArrays*** (page 45), ***AL_SetArraysNam*** (page 41), and ***AL_UpdateArrays*** (page 46).

These commands allow you to implement a dynamic display of data. You should keep in mind that the column number used to refer to a given column, particularly when using any of the multitude of configuration commands, may change as columns are inserted or deleted. In later attempts to configure this column, the new number must be used.

If new arrays of different sizes are to be displayed, then the old arrays must first be removed using ***AL_RemoveArrays****, then the new arrays added with* ***AL_InsArrayNam*** *or* ***AL_SetArraysNam****.*

Modifying Array Elements Procedurally

When the arrays are initially specified via the ***AL_SetArraysNam*** (page 41) or ***AL_InsArrayNam*** (page 43) command, the number of array elements is established for the area. To change the number of elements displayed in the existing arrays, new elements should be added or deleted, and the command ***AL_UpdateArrays*** (page 46) called with *UpdateMethod* set to -2.

If the value or any attribute of an array element is changed or if the number of elements is changed, but the specified arrays are the same, you should instruct AreaList Pro to refresh the area with ***AL_UpdateArrays***.

Specifying the Fields to Display

AreaList Pro uses the new **SubselectionToArray** command in 4D to get the records for display. This command is available beginning with 4D v3.5.3. Therefore fields can not be displayed in an AreaList Pro object when used with an earlier version of 4D.

See “Field and Record Commands” on page 95 for the details on display fields.

Using the AreaEntered and AreaExited Callback Procedures

A “callback” is a global 4D procedure which is executed by an external. AreaList Pro lets you make use of callbacks when entering and exiting an AreaList Pro object. This feature provides you with the ability to enable/disable buttons or other variables depending upon which object is active. See “Redrawing the Display from the Callback Procedure” on page 113 for more information on updating buttons or other variables from a callback procedure. Also, you can call ***AL_GotoCell*** (page 126) from the area entered callback to initiate data entry when the object is entered.

Your callback procedures may use any 4D commands, but can only use the AreaList Pro commands shown in Table 3, “Enterability Commands Allowed from a Callback,” on page 108 and Table 4, “Other AreaList Pro Commands Allowed from a Callback,” on page 108.

Note: ***AL_UpdateArrays*** (page 46) can only be called with *UpdateMethod* equal to -1 from a callback procedure. Please read the section “Modifying Array Elements Procedurally” on page 27 for more information. ***AL_UpdateFields*** (page 101) can only be called with *UpdateMethod* equal to 0 or 1 from a callback procedure.

You should not call any AreaList Pro commands which change the number of displayed columns, their position in the area, or their sorted order.

Executing a Callback Upon Entering an Area

An “area entered” callback procedure is a 4th Dimension procedure called whenever the AreaList Pro object is entered. The area entered callback procedure is specified by passing the procedure name in the *AreaEnteredProc* parameter of ***AL_SetMainCalls*** (page 48). If this parameter is a null string then no procedure will be called.

The area entered callback procedure is passed one parameter by AreaList Pro. This parameter is a long integer that corresponds to the name of the AreaList Pro object on the layout.

You must use the declaration
C_LONGINT($1)

in your callback procedure. Since the parameter, the long integer $1, contains 4D's representation of the AreaList Pro object, it can be used as the first parameter of any AreaList Pro procedure called.

Executing a Callback Upon Exiting an Area

An "area exited" callback procedure is a 4th Dimension procedure called whenever the AreaList Pro object is exited. The area exited callback procedure is specified by passing the procedure name in the *AreaExitedProc* parameter of ***AL_SetMainCalls*** (page 48). If this parameter is a null string then no procedure will be called.

The area exited callback procedure is passed one parameter by AreaList Pro. This parameter is a long integer that corresponds to the name of the AreaList Pro object on the layout.

You must use the declaration

```
C_LONGINT($1)
```

in your callback procedure. Since the parameter, the long integer $1, contains 4D's representation of the AreaList Pro object, it can be used as the first parameter of any AreaList Pro procedure called.

Headers

Column headers are set with ***AL_SetHeaders*** (page 50). If more than one line of text is needed in a column header, the *NumHeaderLines* parameter of ***AL_SetHeight*** (page 90) should be used. Additional space can be added to the height of a header by specifying the *HeaderHeightPad* parameter of this command.

Additional header attributes are specified by using ***AL_SetHdrStyle*** (page 56), ***AL_SetFormat*** (page 52), ***AL_SetForeClr*** (page 71), and ***AL_SetBackClr*** (page 73), for style, justfication, foreground color, and background color, respectively. Display of column headers can be supressed using the *HideHeaders* parameter of ***AL_SetMiscOpts*** (page 66).

Footers

Column footers are set with ***AL_SetFooters*** (page 51). If more than one line of text is needed in a column footer, the *NumFooterLines* parameter of ***AL_SetHeight*** (page 90) should be used.

Additional space can be added to the height of a footer by specifying the *FooterHeightPad* parameter of this command.

Additional footer attributes are specified by using ***AL_SetFtrStyle*** (page 57), ***AL_SetFormat*** (page 52), ***AL_SetForeClr*** (page 71), and ***AL_SetBackClr*** (page 73) for style, justification, foreground color, and background color, respectively. Display of column footers can be controlled using the *ShowFooters* parameter of the ***AL_SetMiscOpts*** (page 66) command. *Column footers are hidden by default, so you must use this command if you wish to display footers.*

Column Widths

Column widths are by default sized automatically, an option which can be overridden with ***AL_SetWidths*** (page 51). Normally, there is no need to use this command, but for details about exceptions to this rule please read “Performance Issues with the Formatting Commands” on page 40.

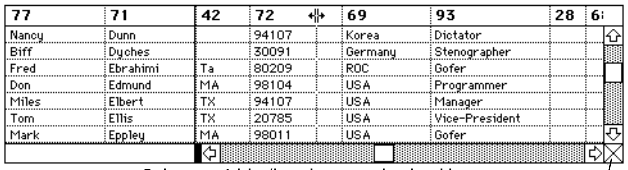
Column widths can be set manually by using ***AL_SetWidths*** (page 51); however, you may want to view the widths generated by AreaList Pro’s automatic column sizing as a good starting reference. The *DisplayPixelWidth* parameter of ***AL_SetColOpts*** (page 62) should be set to 1 to enable this feature, which allows you to toggle between the header text and the column width by clicking on the check box that appears in the bottom right corner of the AreaList Pro object. Additionally, the columns can be resized in the Runtime environment, and the column width values are updated immediately.

When using this feature, you should be sure to enable the display of headers by passing 0 in the *HideHeaders* parameter of ***AL_SetMiscOpts*** (page 66).

AreaList Pro Height

Whenever an array or field command is called, AreaList Pro performs calculations necessary to size the external area based on the size of the external object as drawn on the layout. AreaList Pro will always ensure that only complete rows are displayed in the AreaList Pro area. However, this means that the actual height of the external area as displayed in the User or Runtime Environment may be slightly less than the height in the Layout Editor. This can be a hindrance when you are attempting to align other layout objects with the AreaList Pro object.

To ensure that the AreaList Pro object does not change its size when displayed in the Runtime environment, a tool is available to tell you what size to make the area. To use this tool, first set the *DisplayPixelWidth* parameter of ***AL_SetColOpts*** (page 62) to 1, then click the checkbox as shown.



77	71	42	72	69	93	28	6:
Nancy	Dunn		94107	Korea	Dictator		
Biff	Dyches		30091	Germany	Stenographer		
Fred	Ebrahimi	Ta	80209	ROC	Gofer		
Don	Edmund	MA	98104	USA	Programmer		
Miles	Elbert	TX	94107	USA	Manager		
Tom	Ellis	TX	20785	USA	Vice-President		
Mark	Eppley	MA	98011	USA	Gofer		

Column widths/headers toggle checkbox

The mouse pointer will change from an arrow to a pixel count whenever it is over the list and this option is set. When clicked on a row, this counter will display the necessary height of the AreaList Pro object for that row to be the bottom row displayed. For example, if ten rows are displayed in the area, and you click the seventh row, the number displayed by the pointer will be the height of the object necessary to display exactly seven rows. You can then size the AreaList Pro object in the Design environment using the displayed height. *Please read the section "AreaList Pro Object Dimensions" on page 22 for more information.*

The header size, footer size and the horizontal scroll bar will be taken into account if they are displayed.

Note: This feature is unavailable if enterability can be initiated with a single click.

Column Locking

You can set the lock position using ***AL_SetColLock*** (page 90). ***AL_GetColLock*** (page 153) returns the current position of the column lock. You can also disable the column lock control by using the *AllowColumnLock* parameter of ***AL_SetColOpts*** (page 62).

Rows with Multiple Lines of Text

Row height is determined by a combination of the height of the text line or picture, the number of lines, and any additional padded space. The height of each line of text is determined by the font and point size selected, which are set with ***AL_SetStyle*** (page 58). The number of text lines and the amount of padding are set with the *NumRowLines* and *RowHeightPad* parameters of ***AL_SetHeight*** (page 90). Padded space is the amount of space above and below the text block, (half of the amount above, half below.) All rows will be of the same height.

Color

Column, Header, and Footer Colors

Foreground and background colors can be specified for an AreaList Pro object using ***AL_SetForeClr*** (page 71) and ***AL_SetBackClr*** (page 73). The foreground color can be specified for each column, column header, and column footer, and the background color can be specified for the list area, the header area, and the footer area.

Row-Specific Colors

AL_SetRowColor (page 77) is used to set the foreground and background color of a specified row, and will override any column specification. You can revert to the original column settings by setting the *RowForeColor1* or *RowBackColor1* parameter to the empty string (“”), and the *RowForeColor2* or *RowBackColor2* parameter to -1. Use this command to override all row-specific color settings by passing 0 for the *RowNumber* parameter.

By default, the row color will move with a row if the columns are sorted or a row is dragged. This can be overridden using the *MoveWithData* parameter of ***AL_SetRowOpts*** (page 59).

Cell-Specific Colors

Individual column elements, called cells, can be assigned a unique foreground color and background color. This capability can be used to set negative numbers in red, provide special formatting to show the current selected or enterable cell, and design more attractive and useful lists. These attributes can be set in the **Before** phase, the **During** phase, and either of the AreaList Pro callback procedures (see “Using Callback Procedures During Data Entry” on page 108).

You can use ***AL_SetCellColor*** (page 81) to set the color configuration for an individual cell, a range of cells, or a selection of discontiguous cells. ***AL_GetCellColor*** (page 85) is used to determine any cell-specific colors for a particular cell. ***AL_GetCellColor*** can only determine a color which has been set using the 4D palette of 256 colors, not the AreaList Pro palette.

Use the *MoveWithData* option of ***AL_SetCellOpts*** (page 65) to keep the cell-specific information with a cell when a row or column is dragged to a new location or the list is sorted.

Styles

Column, Header, and Footer Styles

Styles for displayed columns can be set on a column by column basis using ***AL_SetStyle*** (page 58) to set the style for the data, ***AL_SetHdrStyle*** (page 56) to set the header style, and ***AL_SetFtrStyle*** (page 57) to set the footer style. If a 0 is used in the *ColumnNum* parameter, the style will be applied to all columns.

Row-Specific Styles

AL_SetRowStyle (page 75) is used to set the font and style of a specified row, and will override any column specification. You can revert to the original column settings by setting the *StyleNum* parameter to -1. Use this command to override all row-specific style settings by passing 0 for the *RowNum* parameter.

By default, the row style will move with a row if the columns are sorted or a row is dragged. This can be overridden using the *MoveWithData* parameter of ***AL_SetRowOpts*** (page 59).

Cell-Specific Styles

Individual column elements, called cells, can be assigned a unique font and style. This capability can be used to provide special formatting to show the current selected or enterable cell, and design more attractive and useful lists. These attributes can be set in the **Before** phase, the **During** phase, and either of the AreaList Pro callback procedures (see “Using Callback Procedures During Data Entry” on page 108).

You can use ***AL_SetCellStyle*** (page 79) to set the font and style configuration for an individual cell, a range of cells, or a selection of discontiguous cells. ***AL_GetCellStyle*** (page 83) is used to determine any cell-specific formats for a particular cell.

Use the *MoveWithData* option of ***AL_SetCellOpts*** (page 65) to keep the cell-specific information with a cell when a row or column is dragged to a new location or the list is sorted.

Sorting

Sort Buttons

User sorting of the columns via the column header sort buttons is enabled via the *UserSort* parameter of ***AL_SetSortOpts*** (page 69).

Sort Editor

The user can be presented with the AreaList Pro Sort Editor by calling ***AL_ShowSortEd*** (page 157). The prompt at the top of the window defaults to “AreaList™ Pro Sort Editor”, and can be customized using the *SortEditorPrompt* option of ***AL_SetSortOpts*** (page 69). The current sort order of the AreaList Pro area can be displayed when the Sort Editor dialog is presented by setting the *ShowSortOrder* parameter of ***AL_SetSortOpts*** (page 69). If the *AllowSortEditor* option of ***AL_SetSortOpts*** is enabled, the user can invoke the Sort Editor by Command-clicking a column header as described in “Sorting” on page 13.

Procedural Sorting

Multilevel sorting can be performed procedurally on the AreaList Pro columns by using ***AL_SetSort*** (page 86). This command will sort all of the columns in an AreaList Pro area, using up to 15 of them as sort criteria for the multi-level sort. If a column that contains a picture column is passed as one of the sort criteria, that column and all subsequent columns will be ignored. ***AL_GetSort*** (page 149) can be used to retrieve the current sort order of the area, regardless of whether this sort order was established by the user or procedurally.

Sorting When Displaying Fields

Columns containing fields from a related one file will not be sorted when their column header is clicked upon. However, if the *UserSort* option of ***AL_SetSortOpts*** (page 69) is set to 2, “Bypass the user sort buttons”, and the column header of a column containing a field from a related one file is clicked upon, the AreaList Pro area’s script will run, with *ALProEvt* = - 1.

Before AreaList Pro sorts fields (using 4th Dimension’s sorting routines) it turns messages off. If messages were on previously, then AreaList Pro will turn them back on after sorting.

Scrolling

The current scroll position can be set and retrieved using ***AL_SetScroll*** (page 88) and ***AL_GetScroll*** (page 152), respectively.

You can hide either the horizontal or vertical scroll bar, or both, using ***AL_SetScroll*** (page 88). This allows you to construct a grid of cells, providing a different interface from a standard scrolling list.

When a scroll bar is hidden, the user is still able to scroll using the arrow keys or by dragging. You can also set and get the scroll position procedurally.

Selection

Use ***AL_SetEntryOpts*** (page 119) to set the method of selection and data entry. You have extensive control over how the user interacts with a list - a mouse click can select a row, or place the cursor into the cell for data entry. You can also configure the modifier keys (command, shift, option, and control) to control the selection behavior. *Please read the section “Enterability Commands” on page 105 for more information.*

You can configure an AreaList Pro object for no cell selection, single cell selection only, or multiple cell selection, usingthe *CellSelection* parameter of ***AL_SetCellOpts*** (page 65). If you select not to allow cell selection, then the *MultiLines* parameter of ***AL_SetRowOpts*** (page 59) is used to determine the type of row selection — single-line only or multiple-line.

In single line mode, the default configuration requires that one line always be selected. This can be overridden using the *AllowNoSelection* option in ***AL_SetRowOpts*** (page 59), which enables the user to Command-click to deselect the selected row, leaving no rows selected. ***AL_SetRowOpts*** is also used to configure AreaList Pro for single or multiple line selection mode.

You can set the selected rows using ***AL_SetLine*** (page 87) if in single line mode, or ***AL_SetSelect*** (page 87) if in multiple line selection mode.

You can set the selected cells using ***AL_SetCellSel*** (page 84).

When an AreaList Pro object is in cell selection mode, mouse clicks are used to highlight cells rather than rows. If multiple cell selection is enabled using ***AL_SetCellOpts*** (page 65), then the user can shift-click and command-click to select multiple cells. Discontiguous (non-adjoining) selections are allowed. *When an AreaList Pro object is in cell selection mode, it is always possible that no cells are selected.*

AL_SetCellSel (page 84) is used to select cells procedurally, and can select a single cell, a range of cells, or a list of cells. You can determine the selected cells using ***AL_GetCellSel*** (page 151).

When the user scrolls an AreaList Pro object that is in cell selection mode using the arrow keys or keyboard type-ahead, the list will scroll, but the cell selection will not change.

Note: Row dragging is disabled when an AreaList Pro object is in cell selection mode.

The enterability options set with ***AL_SetEntryOpts*** (page 119) are fully supported when an AreaList Pro object is in cell selection mode.

If an AreaList Pro object is in multi-cell selection mode, the Edit menu Select All command is enabled.

Clipboard

The data copied to the clipboard can be formatted using ***AL_SetCopyOpts*** (page 68). This command allows you to specifiy the field and record delimiters copied with the data, and whether any Hidden Column data should be copied to the clipboard.

The Edit menu Copy command is disabled when an AreaList Pro object has been set to allow cell selection using ***AL_SetCellOpts*** (page 65).

Copying rows to the clipboard will not be allowed when displaying fields. The Copy menu item will be disabled when fields are displayed. See “Field and Record Commands” on page 95 for more information about displaying fields.

Picture Columns

AreaList Pro supports the display of picture columns. The *Format* parameter of ***AL_SetFormat*** (page 52) will cause the picture to be displayed in one of several ways:

- truncated and justified to the upper left of the cell
- truncated and centered in the cell
- scaled to fit the cell
- scaled proportionally to fit the cell.

The *UsePictHeight* parameter of this command will tell AreaList Pro whether to use a picture's original height, which is stored with the picture, when calculating the row height for the AreaList Pro area. If you choose not to use the picture's height in the row height calculation and additional space is needed to display the picture, the *NumRowLines* parameter of ***AL_SetHeight*** (page 90) should be used to increase the row height.

Saving and Restoring Configuration Information

Once an area is configured using the many commands provided by AreaList Pro, the state of that area can be saved in a picture variable or field using ***AL_SaveData*** (page 47). ***AL_RestoreData*** (page 49) then allows you to restore the configuration to this or another AreaList Pro area at some time in the future, saving much of the effort required to configure multiple AreaList Pro objects. The information saved in the picture is that controlled by the AreaList Pro commands shown in Table 1.

Table 1: Configuration Commands Used by *AL_SaveData*

AL_SetWidths	***AL_SetHeaders***	***AL_SetFormat***
AL_SetStyle	***AL_SetHdrStyle***	***AL_SetForeClr***
AL_SetBackClr	***AL_SetDividers***	***AL_SetHeight***
AL_SetRowOpts	***AL_SetColOpts***	***AL_SetMiscOpts***
AL_SetCopyOpts	***AL_SetSortOpts***	***AL_SetColLock***
AL_SetFooters	***AL_SetFtrStyle***	***AL_SetEnterable****
AL_SetFilter	***AL_SetCallbacks***	***AL_SetEntryOpts***
AL_SetEntryCtls	***AL_SetCellOpts***	***AL_SetDrgSrc***
AL_SetDrgDst	***AL_SetDrgOpts***	***AL_SetMainCalls***
AL_SetWinLimits		

* The *PopupArray* parameter of this command will not be saved.

Note: The purpose of this command is to restore an AreaList Pro area to its original configuration, and alterations to the area caused by the user will not be reflected in the saved picture.

The configuration information saved in this picture also can be used to configure a PrintList Pro v3.0 area using ***PL_RestoreData*** (please see the PrintList Pro Reference for more information). Much of the saved AreaList Pro information listed above does not apply to a PrintList Pro area and will be ignored. The information controlled by the commands listed below will be used to configure the PrintList Pro area:

Table 2: Configuration Commands Which Can be Used to Configure a PrintList Pro Object

AL_SetWidths	***AL_SetHeaders***	***AL_SetFormat***
AL_SetStyle	***AL_SetHdrStyle***	***AL_SetForeClr***
AL_SetBackClr	***AL_SetDividers***	***AL_SetHeight***

Note: Any part of this information concerning AreaList Pro footers is ignored by PrintList Pro.

Changing Layout Pages

If an AreaList Pro object is displayed on a multipage layout, you must inform it when the user changes to another layout page. This is done by calling the following command whenever the layout is changed to a different page:

```
AL_SetScroll(eList;0;0)   `inform AreaList Pro object that page is being
                          changed
```

where eList is the name of the AreaList Pro object on the page you are leaving. If this is not done the AreaList Pro object’s scroll bars may be active on another page.

If the Drag and Drop feature of AreaList Pro is used on a multi-page layout, a similar action must be performed. When pages are changed in the layout, you must ensure that Drag and Drop is enabled only for AreaList Pro areas on the current page (if this feature is desired), and that Drag and Drop is disabled for any AreaList Pro areas on other pages. *Please read the section “AreaList Pro on Multi-Page Layouts” on page 132 for more information.*

AL_SetDropDst (page 156) should be used to disable a DropArea on the current page when moving to a different layout page. *Please read the section "DropArea Objects on a MultiPage Layout" on page 155 for more information.*

Using AreaList Pro on a Resizable Window

An AreaList Pro object and its window may be made resizable using ***AL_SetWinLimits*** (page 91). Only one resizable AreaList Pro object may be placed on a layout. Other objects (4D variables, AreaList Pro objects, etc.) may be placed to the left or above this resizable object, but no objects may be placed to the right or below this object.

The AreaList Pro object is not really resizable. It appears this way because AreaList Pro draws its scroll bars at the right and bottom edges of the window instead of the right and bottom edges of the area when this option is enabled.

Creating a Resizable AreaList Pro Area

The following steps are necessary to produce a resizable AreaList Pro object:

1 Create an AreaList Pro object on a layout. Make sure that the area extends well beyond (in the 1200 to 1300 pixel range) the right and bottom edges of the layout.
2 Call ***AL_SetWinLimits*** (page 91) to enable resizing and to specify the minimum and maximum width and height of the window.
3 The vertical scroll bar must be shown in the AreaList Pro object for the size box to be drawn properly. The horizontal scroll bar may be shown or hidden. See "Scrolling" on page 35 for more information.
4 Make certain that the window type used does not contain a size (grow) box. The AreaList Pro object will draw its own size box. The window types 4 or 12 are recommended. If a window type with a size box is chosen, AreaList Pro cannot limit the size of the window to the minimum or maximum settings.
5 Within the script of the AreaList Pro object, in the **During** phase, if *ALProEvt* is equal to -9 (the AreaList Pro object was resized), call ***AL_DoWinResize*** (page 93). This command allows AreaList Pro to change the window size to what the user selected via the size box.

Note: If the user clicks in the zoom box (in a type 12 window), no interaction is necessary. The window and the AreaList Pro object *will be resized automatically.*

Note: When the AreaList Pro object is resized to a smaller width, the column sizes (see “Column Widths” on page 11) and the column lock will be adjusted accordingly. Consequently you should set the minimum width and height to large enough values to minimize this effect.

Performance Issues with the Formatting Commands

AreaList Pro uses an algorithm to automatically size the columns. Because of this, there is usually no need to use ***AL_SetWidths*** (page 51) to manually size a column prior to displaying a list. However, if the number of items in the list is very large (more than 2,000 items with many columns), then the list might take one or two seconds to display, due to the automatic sizing calculation. If this is the case, using ***AL_SetWidths*** will improve the display time of the list. Text and string columns will take the longest to automatically size. Since you can use ***AL_SetWidths*** on just some of the columns, if you are displaying very large arrays, but only one is text or string, you could use the ***AL_SetWidths*** command on just the text or string column, and let AreaList Pro automatically calculate the other column widths. To determine the optimum width for a column, you can display the pixel widths of columns in the headers during your design process, and then use ***AL_SetWidths*** to set the width. See “Column Widths” on page 30 and the ***AL_SetWidths*** definition on page 51 for more information.

Note: When AreaList Pro display fields, the automatic column sizing algorithm uses only the first 20 records (or less, if the selection contains less than 20 records) in the selection. These records are always read regardless of whether the columns are automatically or manually sized. Therefore there is no performance penalty using the automatic column sizing algorithm when displaying fields. See “Field and Record Commands” on page 95 for more information about displaying fields.

AL_SetFormat (page 52) does not affect the performance of AreaList Pro, regardless of the size of the columns being displayed. This is because AreaList Pro is using 4D’s array data directly, and as the list is scrolling, the formatting is being done “on-the-fly.”

Sorting the columns will have the greatest impact on the time required for AreaList Pro to be displayed in the **Before** phase or updated in the **During** phase. If you will be displaying many large columns, you can reduce the display time by turning off the *SortInDuring* option using ***AL_SetSortOpts*** (page 69).

Commands

%AreaListPro

%AreaListPro is the command used to identify the AreaList Pro external area when you create an external area object on a layout. This command is only used in the object definition for an AreaList Pro object, and should never be used as a command in a script or procedure.

AL_SetArraysNam

AL_SetArraysNam*(AreaName; ColumnNum; NumArrays; Array1; … ; ArrayN)* ➞ *ErrorCode*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
ColumnNum	integer	column at which to set the first array
NumArrays	integer	number of arrays to set (up to 15)
Array	string	name of 4D array
ErrorCode	integer	error code

AL_SetArraysNam tells AreaList Pro what arrays to display. Up to fifteen arrays can be set at a time. *Any 4D array type can be used except pointer and two-dimensional arrays.* There are three very important points to note about this command:

- This command *must* be called first, before any of the other commands, in both the **Before** and **During** phases.
- The columns *must* be added in sequential order, unless the particular column has already been added. In other words, to set 30 arrays, you must set arrays 1 through 15 prior to setting arrays 16 through 30.
- All arrays set with this command *must* have the same number of elements as each other and as any other arrays previously set.

AL_SetArraysNam may be called in the **Before** phase to initially set the arrays to be displayed. Since AreaList Pro can display up to 255 arrays, this command may have to be used more than once. However, it is not mandatory to set any arrays in the **Before** phase; in that case the area on the layout where AreaList

Pro is defined will be blank.

You can pass Process arrays and Interprocess arrays to AreaList Pro, but not Local arrays (a local array has a name that starts with a "$" character; an interprocess array has a name that starts with a "◊" character on the Macintosh and the "<>" characters on Windows)).

One dimension of a two-dimensional array may be passed in the *Array1; … ; ArrayN* parameters. For example: “My2DArray{1}” may be passed as Array1.

AL_SetArraysNam replaces the ***AL_SetArrays*** command (pre-version 5). *Please read the section “Array Setup” on page 159 for more information.*

ColumnNum — Integer. This parameter specifies the column number to set the first array being passed by this call of ***AL_SetArraysNam***.

NumColumns — Integer. This parameter specifies the number of columns being set with this call to ***AL_SetArraysNam***.

ErrorCode — Integer. The possible values are:

Valu e	Error Code	Action
0	No error	n/a
1	Not an array	check to make sure all arrays are correctly typed
2	Wrong type of array	pointer and two-dimensional arrays are not allowed
3	Wrong number of rows	make sure that all arrays have the same number of elements
4	Maximum number of arrays exceeded	100 arrays is the maximum
5	Not enough memory	Increase 4D’s RAM partition, or change your approach to use fewer or smaller arrays

AL_SetArraysNam may be called in the **During** phase to set arrays to be displayed or to replace arrays that are already displayed.

Examples:

```
  `AreaList Pro eNameList script
Case of
  :(Before)
    SELECTION TO ARRAY([Contacts]FN;aFN;[Contacts]LN;aLN;[Con-
                tacts]City;aCity;[Contacts]State;aState)  `load the arrays
    $Error:=AL_SetArraysNam(eNameL-
                ist;1;4;"aFN";"aLN";"aCity";"aState")  `starting at column 1,
                set 4 arrays
  :(During)
    Case of
      :(ALProEvt=2)  `user double-clicked
      :(ALProEvt=1)  `user single-clicked
      :(ALProEvt=-1)  `user sorted
End case

  `set up the eList AreaList Pro object with 25 arrays
  `two calls must be made since only 15 arrays can be passed each time
$Error:=AL_SetArraysNam(eList;1;15;"array1";"array2";"array3";"array4";"arr
                ay5";"array6";
                "array7";"array8";"array9";"array10";"array11";"array12";"array
                13"; "array14";"array15")
$Error:=AL_SetArraysNam(eList;16;10;"array16";"array17";"array18";"array1
                9";"array20"; "array21";"array22";
                "array23";"array24";"array25")
```

AL_InsArrayNam

AL_InsArrayNam*(AreaName; ColumnNum; NumArrays; Array1; … ; ArrayN) ➞ ErrorCode*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
ColumnNum	integer	column at which to insert the first array
NumArrays	integer	number of arrays to insert (up to 15)
Array	array	4D array
ErrorCode	integer	error code

AL_InsArrayNam functions the same as ***AL_SetArraysNam*** (page 41), except that the arrays are inserted before *ColumnNum*.

All subsequent columns will maintain their settings. In other words, any header text, column styles, etc. will stay with their corresponding array.

Up to fifteen arrays can be set at a time. Any 4D array type can be used except pointer and two-dimensional arrays. There are three very important points to note about this command:

- This command (or ***AL_SetArraysNam***) *must* be called first, before any of the other commands, in both the **Before** and **During** phases.

- The columns *must* be added in sequential order, unless the particular column has already been added. In other words, to set 30 arrays, you must set arrays 1 through 15 prior to setting arrays 16 through 30.
- All arrays set with this command *must* have the same number of elements as each other and as any other arrays previously set.

AL_InsArrayNam may be called in the **Before** phase to initially set the arrays to be displayed (although you will usually use ***AL_SetArraysNam*** in the **Before** phase). Since AreaList Pro can display up to 255 arrays, this command may have to be used more than once. However, it is not mandatory to set any arrays in the **Before** phase; in that case the area on the layout where AreaList Pro is defined will be blank.

You can pass Process arrays and Interprocess arrays to AreaList Pro, but not Local arrays (a local array has a name that starts with a "$" character; an interprocess array has a name that starts with a "◊" character on the Macintosh and the "<>" characters on Windows)).

One dimension of a two-dimensional array may be passed in the *Array1; … ; ArrayN* parameters. For example: “My2DArray{1}” may be passed as Array1.

AL_InsArrayNam replaces the ***AL_InsertArrays*** command (pre-version 5). *Please read the section “Array Setup” on page 159 for more information.*

ColumnNum — Integer. This parameter specifies the column number to set the first array being passed by this call of ***AL_InsArrayNam***.

NumColumns — Integer. This parameter specifies the number of columns being set with this call to ***AL_InsArrayNam***.

ErrorCode — Integer. The possible values are:

Value	Error Code	Action
0	No error	n/a
1	Not an array	check to make sure all arrays are correctly typed

Value	Error Code	Action
2	Wrong type of array	pointer and two-dimensional arrays are not allowed
3	Wrong number of rows	make sure that all arrays have the same number of elements
4	Maximum number of arrays exceeded	100 arrays is the maximum
5	Not enough memory	Increase 4D's RAM partition, or change your approach to use fewer or smaller arrays

Example:

```
$Error:=AL_InsArrayNam(eList;4;3;"aFN";"aLN";"aComp")  `starting at column 4, insert 3 arrays
```

AL_RemoveArrays

AL_RemoveArrays*(AreaName; ColumnNum; NumArrays)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
ColumnNum	integer	column at which to remove the first array
NumArrays	integer	number of arrays to remove (up to 255)

AL_RemoveArrays is used to remove arrays from AreaList Pro. *NumArrays*, beginning at *ColumnNum*, will be removed from the list.

All subsequent columns will maintain their settings. In other words, any header text, column styles, etc. will stay with their corresponding array.

Examples:

```
AL_RemoveArrays(eList;8;4)  `starting at column 8, remove 4 arrays
AL_RemoveArrays(eList;1;20)  `remove all 20 arrays
```

AL_UpdateArrays

AL_UpdateArrays*(AreaName; UpdateMethod)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
UpdateMethod	integer	method to use to update the AreaList Pro object

AL_UpdateArrays is used to update AreaList Pro. Use this command whenever any elements of the arrays being displayed are changed (elements added, deleted, or modified), but the arrays themselves remain the same.

AL_UpdateArrays *must* be called *after* modifying the arrays and *before* any other setup commands (sorting, formatting, etc.).

UpdateMethod— integer. This parameter tells AreaList Pro how to update the AreaList Pro object *AreaName*.

Value	Description	When to Use
-2	Rescan all arrays and recalculate all applicable heights, widths, and other related values. The scroll position, and row or cell selection will be reset. This value is converted to -1 if passed from a callback procedure.	If column or row resizing is necessary, or you have added or removed elements to any of the displayed arrays. Also if you show or hide either scroll bar, the headers, or footers, or add or remove arrays.
-1	Refresh the AreaList Pro object, but don't recalculate any values.	The AreaList Pro object needs to be updated because of changes to an array element's contents, or formatting changes to colors, styles, etc. This value should only be used when no column or row resizing is necessary, since formatting, styles, or an element's new contents could affect a column width or row height.

*Note: You may only pass a value of -1 for UpdateMethod when calling **AL_UpdateArrays** from a callback procedure. Please read the section "Using Callback Procedures During Data Entry" on page 108 for more information.*

Example:

```
If(During)
    `any action which modifies an array element value, or changes a configuration attribute
    `must include updating the AreaList Pro object
  AL_UpdateArrays(eList;-2)
End if

  `bDeleteLines button script
  `This example shows how to delete elements from displayed
  `arrays and how to update AreaList Pro
  `The routine deletes selected lines in an AreaList Pro object named eList.
  `eList is configured for multiple-line selection,
  `and it is displaying three arrays: aFN, aLN, aComp
ARRAY INTEGER(aLines;0)  `create an integer array with a size of zero
$OK:=AL_GetSelect(eList;aLines)  `get the lines selected by the user, put into aLines array
If($OK=1)  `enough RAM was available to resize the aLines array
  For($i;Size of array(aLines);1;-1)  `start at the end of the array and go to top
    DELETE ELEMENT(aFN;aLines{$i})  `delete the selected element from the three arrays
    DELETE ELEMENT(aLN;aLines{$i})
    DELETE ELEMENT(aComp;aLines{$i})
  End for
  AL_GetScroll(eList;vVert;vHoriz)  `get current scroll position
  AL_UpdateArrays(eList;-2)  `update the AreaList Pro object
  AL_SetScroll(eList;vVert;vHoriz)  `reset scroll position so it doesn’t change
End if
```

AL_SaveData

AL_SaveData*(AreaName; SavePict)* ➞ *ResultCode*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
SavePict	picture	location to save the setup data
ResultCode	integer	identifies result condition

AL_SaveData is used to save the current configuration of the AreaList Pro object. See “Saving and Restoring Configuration Information” on page 37 for more information, including what information is saved, and how this information can be used to configure both AreaList Pro and PrintList Pro objects

ResultCode — Integer, 0 or 1.

1 the command was successful and the setup information was saved

0 there was not enough memory to save the configuration

AL_SetMainCalls

AL_SetMainCalls *(AreaName; AreaEnteredProc; AreaExitedProc)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
AreaEnteredProc	string	4D procedure called when object is entered
AreaExitedProc	string	4D procedure called when object is exited

AL_SetMainCalls is used to set callback procedures that are used when entering and exiting the AreaList Pro object.

AreaEnteredProc — String. This procedure will be called whenever the AreaList Pro object is entered. If this is the null string then no procedure will be called.

The *AreaEnteredProc* is passed one parameter. This parameter is a long integer that corresponds to the name of the AreaList Pro object on the layout.

Note: If the AreaList Pro object is the first object in the entry order, when the layout containing the AreaList Pro object is first opened, the *AreaEnteredProc* will not be called. This is because 4D gives the event to AreaList Pro (to inform it that it is to be the active object when the layout is opened) prior to the execution of the layout's **Before** phase. If you want to take action based upon this active object, then call the *AreaEnteredProc* from the **Before** phase in your 4D code.

AreaExitedProc — String. This procedure will be called whenever the AreaList Pro object is exited. If this is the null string then no procedure will be called.

The *AreaExitedProc* is passed one parameter. This parameter is a long integer that corresponds to the name of the AreaList Pro object on the layout.

Some of the uses of these callbacks are as follows:

- Enabling buttons or other variables that pertain to the AreaList Pro object from the *AreaEnteredProc*. You must use interprocess global buttons or variables and call **CALL PROCESS** (-1) to update them.
- Disabling buttons or other variables that pertain to the

AreaList Pro object from the *AreaExitedProc*. You must use interprocess global buttons or variables and call **CALL PROCESS** (-1) to update them.

- Call ***AL_GotoCell*** (page 126) from the *AreaEnteredProc* to initiate data entry when the object is entered.

Example:

```
   ` set up area entered and area exited callbacks
AL_SetMainCalls (eList;"AreaEntrProc";"AreaExitProc")


   ` AreaEntrProc, area entered callback procedure
C_LONGINT($1)

AL_GotoCell ($1;1;1)   ` Initiate data entry on the first cell in the first column

ENABLE BUTTON(◊bChangeSub)
ENABLE BUTTON(◊bAltRowBkd)
CALL PROCESS(-1)


   ` AreaExitProc, area exited callback procedure
C_LONGINT($1)

DISABLE BUTTON(◊bChangeSub)
DISABLE BUTTON(◊bAltRowBkd)
CALL PROCESS(-1)
```

AL_RestoreData

AL_RestoreData*(AreaName; RestorePict) ➞ ResultCode*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
SavePict	picture	location of saved setup data
ResultCode	integer	identifies result condition

AL_RestoreData is used to restore a saved configuration to an AreaList Pro object. See “Saving and Restoring Configuration Information” on page 37 and the ***AL_SaveData*** command (page 47) for more information.

ResultCode — Integer, 0 or 1.

1	the command was successful and the setup information was restored
0	the information in the picture was not valid AreaList Pro or PrintList Pro information

AL_SetHeaders

AL_SetHeaders*(AreaName; ColumnNum; NumHeaders; Header1; … ; HeaderN)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
ColumnNum	integer	column at which to set the first header
NumHeaders	integer	number of headers to set (up to 15)
Header	string	value to display in column header

AL_SetHeaders is used to specify the value to display in the header for each column. Up to fifteen headers can be set at a time.

The size of the header value is used by the automatic column sizing algorithm. If you are displaying a fixed-string array with an element size of 2 characters, the column will be very narrow, unless you specify a header which contains several characters. For example, states are usually stored in a database as a two-character alpha, and you would probably display them directly or load them into a string array sized for two-characters length. But if you specify a header of “State” the column will be sized about two and a half times wider. If the header length is less than the values being displayed in the column, then the header length will not affect the column width.

A, B, C, etc. will be displayed in the headers if ***AL_SetHeaders*** is not used. The ***AL_SetHeaders*** command can be used in both the **Before** phase and **During** phase of a script or procedure.

Examples:

```
$Error:=AL_SetArraysNam(eNameList;1;4;"aFN";"aLN";"aCity";"aState")
AL_SetHeaders(eNameList;1;4;"First Name";"Last Name";"City";"State")

$Error:=AL_SetArraysNam(eNames;1;2;"aFN";"aLN")
AL_SetHeaders(eNames;1;2;Fieldname([People]FirstName);Field-
                  name([People]LastName))
```

Footer	string	value to display in column footer

AL_SetFooters

AL_SetFooters*(AreaName; ColumnNum; NumFooters; Footer1; … ; FooterN)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
ColumnNum	integer	column at which to set the first footer
NumFooters	integer	number of footers to set (up to 15)

AL_SetFooters is used to specify the value to display in the footer for each column. Up to fifteen footers can be set at a time. The *ShowFooters* option of ***AL_SetMiscOpts*** (page 66) *must* be enabled.

The size of the footer value is used by the automatic column sizing algorithm the same way that the header for a column is used. For more information, See "***AL_SetHeaders***" on page 50.

Nothing will be displayed in the footer area if ***AL_SetFooters*** is not used. ***AL_SetFooters*** can be used in both the **Before** phase and **During** phase of a script or procedure.

Example:

```
For($i;1;Size of array(aSalary))
  $Total := $Total+aSalary{$i}
End for
AL_SetFooters(eEmpList; 3; 1; String ($Total))
```

AL_SetWidths

AL_SetWidths*(AreaName;ColumnNum;NumWidths;Width1; … ; WidthN)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
ColumnNum	integer	column at which to set the first width
NumWidths	integer	number of widths to set (up to 15)
Width	integer	pixel width of column

AL_SetWidths is used to set the pixel width for one or more columns. Up to fifteen widths can be set at a time. A Width of zero forces a column to be sized automatically based on its data type.

A column cannot be less than 3 pixels wide. If you pass a value of less than 3 but greater than zero, AreaList Pro will ignore it and use 3. AreaList Pro will not let a column be wider than the width of the list area minus 20.

If not called, the default width for all columns is determined based on the type of array or field displayed in the column and the header for the column.

AL_SetWidths can be used in both the **Before** phase and **During** phase of a script or procedure.

Example:

```
$Error:=AL_SetArraysNam(eNames;1;5;"aFN";"aLN";"aCity";"aState";"aZip"
                    )
AL_SetWidths(eNames;1;5;150;50;0;100;0)  `0 forces autosizing for that col-
                    umn!
```

AL_SetFormat

AL_SetFormat*(AreaName;ColumnNum;Format;ColumnJust;HeaderJust;FooterJust;*
UsePictHeight)

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
ColumnNum	integer	column number to format
Format	string	format to use
ColumnJust	integer	justification for column list items
HeaderJust	integer	justification for column header
FooterJust	integer	justification for column footer
UsePictHeight	integer	use the picture height in the row height calculation

AL_SetFormat is used to control the format and justification of a column being displayed. You can control the format of string, integer, long integer, real, date, boolean, and picture columns with the *Format* parameter. Time values can be formatted also, since they use long integer arrays. Any valid 4D format, including custom formats created in the Design Environment, may be used with these column types, *except for string arrays*. Text columns cannot be formatted.

Additionally, null time and date values can be set to display a blank by appending a dash character (“-”) to the *Format* string parameter.

The defaults for the different column types are:

Column Type	Format
String	none
Integer	“##,##0”
Long Integer	“#,###,##0”
Real	“#,###,##0.00”
Boolean	“True;False”
Date	“0”
Picture	“0”

Format (for string arrays) — String. Any formatting characters supported for 4D are allowed. *Pre-defined Styles (i.e., those saved in the Design Environment) are not allowed.*

Format (for text columns) — *Not Supported*.

Format (for numeric columns) — String. See the 4D command **String** in the 4D Language Reference for the possible values. Any valid 4D numeric format may be used.

Format (for bolean columns) — String. The string contains two formats, one for the True value, the other for the False value, separated by a semicolon. Examples: “Male;Female” and “Macintosh;Windows.”

Format (for date columns) — String. See the 4D command **String** in the 4D Language Reference for the possible values. Any valid 4D date format may be used. Examples: “0” or “3” are valid formats.

Format	Example
0	4/19/95 (default)
1	4/19/95
2	Wed, Apr 19, 1995
3	Wednesday, April 16, 1995
4	04/19/95 or 04/19/1895

Format	Example
5	April 19, 1995
6	Apr 19, 1995

Format (for time columns) — String. See the 4D **String** command in the 4D Language Reference, and the 4D Design Reference discussion of formatting for the possible values. There are no time arrays in 4D as such, they are in reality long integer arrays. These arrays are displayed as time values by using the proper format. The format is the two character sequence “&/” followed by the number given in the discussion of the **String** command. For example, one proper format for a time array would be “&/2”.

Format	Example
1	01:02:03
2	01:02
3	1 hour 2 minutes 3 seconds
4	1 hour 2 minutes
5	1:02 AM

Format (for picture columns) — String.

0 the picture will be truncated, if necessary, and justified to the upper left (default)
1 the picture will be truncated, if necessary, and centered in the cell
2 the picture will be scaled to fit the cell.
3 the picture will be scaled to fit the cell, and remain proportional to its original size

ColumnJust, *HeaderJust*, and *FooterJust* — Integer. The justification for a column, its header, and its footer can be controlled independently. The possible values are:

Value	Justification
0	Default
1	Left
2	Center
3	Right

By default, headers are left justified, unless the column elements are center justified. In that case, the header will default to center justification.

The default footer justification corresponds to the column justifications, which for the different column types are:

Column Type	Default Column Justification
Integer	right
LongInteger (including Time)	right
Real	right
Boolean	left
Date	right
String	left
Text	left
Picture	n/a - see the *Format* parameter

The *ColumnJust* parameter is ignored for picture columns. Use the *Format* parameter to justify picture columns.

UsePictHeight — Integer, 0 or 1.

1 use height of the largest picture calculating the row height
0 ignore the picture height when calculating the row height (default)

If the column *ColumnNum* does not have a picture column, this parameter will be ignored.

AL_SetFormat can be used in both the **Before** phase and **During** phase of a script or procedure.

Examples:

```
  `format a real column (3rd column), default column
  `justification,  center header justification, and default footer justification
AL_SetFormat(names;3;"$###,###.00";0;2;0;0)

  `format a string (2nd column), default
  `column justification and default header justification, center footer justification
```

```
AL_SetFormat(eContacts;2;"(###) ###-####";0;0;2;0)

  `format a boolean column (4th column), right column
  `justification and left header justification
AL_SetFormat(eList;4;"Male;Female";3;1;0;0)

  `format style 3 for a date column, default justification (5th column)
  `default column, header, and footer justification
AL_SetFormat(eNames;5;"3")

  `format style 2 for a time column, right justification for header and column
                    (7th column)
AL_SetFormat(eList;7;"&/2";3;3;0;0)

  `custom format style, default justification for column, center
  `header (5th column)
AL_SetFormat(eNames;5;"|Dollars";0;2;0;0)

  `Scale picture column to fit proportionally (1st column)
  `use default header and footer justification, and use picture size in row
                    height calculation
AL_SetFormat(ePeople;1;"3";0;0;0;1)
```

AL_SetHdrStyle

AL_SetHdrStyle*(AreaName;ColumnNum;FontName;Size;StyleNum)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
ColumnNum	integer	number of column
FontName	string	name of the font
Size	integer	size of the font
StyleNum	integer	style of the font

AL_SetHdrStyle is used to control the appearance of the AreaList Pro column headers. The columns can be controlled individually or as a group.

ColumnNum — Integer. This parameter specifies what column header to apply the style to. Use a value of zero (0) to apply the parameters to all columns.

StyleNum — Integer. The *StyleNum* is a Macintosh font style code. By adding the codes together, you can combine styles. The numeric codes for *StyleNum* are shown below.

Style	Number
Plain	0

Style	Number
Bold	1
Italic	2
Underline	4
Outline	8
Shadow	16
Condensed	32
Extended	64

FontName — String. Use this parameter to specify the font for the specified *ColumnNum*. If not called, or the specified *FontName* is not found, the header(s) will be displayed in Geneva 12 point plain. If the font specified by *FontName* is not installed, then Geneva will be used.

AL_SetHdrStyle can be used in both the **Before** phase and **During** phase of a script or procedure.

Examples:

AL_SetHdrStyle(eList;1;"Geneva";12;1) `Geneva 12 point bold, column 1
AL_SetHdrStyle(Names;3;"New York";12;3) `New York 12 point bold italic, column 3
AL_SetHdrStyle(Names;0;"Palatino";10;3) `Palatino 10 point bold italic, all columns

AL_SetFtrStyle

AL_SetFtrStyle*(AreaName;ColumnNum;FontName;Size;StyleNum)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
ColumnNum	integer	number of column
FontName	string	name of the font
Size	integer	size of the font
StyleNum	integer	style of the font

AL_SetFtrStyle is used to control the appearance of the AreaList Pro column footers. The columns can be controlled individually or as a group.

ColumnNum — Integer. This parameter specifies what column footer to apply the style to. Use a value of zero (0) to apply the parameters to all columns.

StyleNum — Integer. The *StyleNum* is a Macintosh font style code. By adding the codes together, you can combine styles. The numeric codes for *StyleNum* are shown below.

Style	Number
Plain	0
Bold	1
Italic	2
Underline	4
Outline	8
Shadow	16
Condensed	32
Extended	64

FontName — String. Use this parameter to specify the font for the specified *ColumnNum*. If not called, or the specified *FontName* is not found, the header(s) will be displayed in Geneva 12 point plain. If the font specified by *FontName* is not installed, then Geneva will be used.

AL_SetFtrStyle can be used in both the **Before** phase and **During** phase of a script or procedure.

AL_SetStyle

AL_SetStyle*(AreaName;ColumnNum;FontName;Size;StyleNum)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
ColumnNum	integer	number of column
FontName	string	name of the font
Size	integer	size of the font
StyleNum	integer	style of the font

AL_SetStyle is used to control the appearance of the AreaList Pro columns. The columns can be controlled individually or as a group.

ColumnNum — Integer. This parameter specifies what column to apply the style to. Use a value of zero (0) to apply the parameters to all columns.

StyleNum — Integer. The *StyleNum* is a Macintosh font style code. By adding the codes together, you can combine styles. The numeric codes for *StyleNum* are shown below.

Style	Number
Plain	0
Bold	1
Italic	2
Underline	4
Outline	8
Shadow	16
Condensed	32
Extended	64

FontName — String. Use this parameter to specify the font for the specified *ColumnNum*. If not called, or the specified *FontName* is not found, the header(s) will be displayed in Geneva 10 point plain. If the font specified by *FontName* is not installed, then Geneva will be used.

AL_SetStyle can be used in both the **Before** phase and **During** phase of a script or procedure.

Examples:

AL_SetStyle(eNames;0;"Geneva";9;0) `Geneva 9 Plain, all columns
AL_SetStyle(eList;4;"Helvetica";12;32) `Helvetica 12 point Condensed, 4th column
AL_SetStyle(eNames;1;"Times";9;1) `Times 9 point bold, 1st column

AL_SetRowOpts

AL_SetRowOpts*(AreaName;MultiLines;AllowNoSelection;DragLine;AcceptDrag;MoveWithData;DisableRowHighlight)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
MultiLines	integer	single or multiple-line selection
AllowNoSelection	integer	allow no lines to be selected in single line mode
DragLine	integer	drag a line to this or another object
AcceptDrag	integer	accept drag from another AreaList Pro object
MoveWithData	integer	move row style and color with row
DisableRowHighlight	integer	disable highlighting of selected rows

AL_SetRowOpts is used to control several AreaList Pro options pertaining to rows.

Note: the DragLine and AcceptDrag parameters are ignored when using the Macintosh Drag Manager routines provided in AreaList Pro v5.1 and later. Please read “Dragging Commands” on page 129 and “Obsolete Dragging Commands” on page 177 for more information.

MultiLines — Integer, 1 or 0.

1 allow the user to command-click, shift-click, or drag to select multiple lines
0 allow only one line to be selected (default).

In multi-line mode, no lines are initially selected unless ***AL_SetSelect*** (page 87) is used. In single-line mode, the first line is selected unless ***AL_SetLine*** (page 87) is used.

AllowNoSelection — Integer, 1 or 0.

1 the user can command-click to deselect a line in single line mode
0 the user can not deselect a line (default)

Regardless of the value of *AllowNoSelection*, ***AL_SetLine*** (page 87) can be used with the *LineNum* parameter set to 0 to set the selection to no lines.

DragLine — Integer, 0 to 6. The default is 0.

0 do not allow a line to be dragged (default)

Values 1, 2, and 3 enable a line to be dragged while the Option key is pressed:

1 allow a line to be dragged within, but not out of the AreaList Pro object.
2 allow a line to be dragged out of, but not within the AreaList Pro object.
3 allow a line to be dragged both within and out of the AreaList Pro object.

Values 4, 5, and 6 enable a line to be dragged without any modifier key:

4 allow a line to be dragged within, but not out of the AreaList Pro object
5 allow a line to be dragged out of, but not within the AreaList Pro object

6 allow a line to be dragged both within and out of the AreaList Pro object

If a line is dragged without any modifier key, dragging to select multiple lines will not work.

If the line is dragged to another position within the list, AreaList Pro will automatically rearrange all of the columns. If the line is dragged out of the list to another AreaList Pro object, it is up to you to remove and insert row(s) as necessary. See “***AL_GetDragLine***” on page 182 for more information.

AcceptDrag — Integer, 1 or 0.

1 this AreaList Pro object will accept a line dragged from another AreaList Pro object
0 this AreaList Pro object will not accept a line(default)

MoveWithData — Integer, 1 or 0. This parameter is used with ***AL_SetRowStyle*** (page 75) and ***AL_SetRowColor*** (page 77).

1 the row style and color information will move with the row whenever the AreaList Pro object is sorted or a row is dragged within the list (default)
0 the row style and color information will not move with the row.

DisableRowHighlight — Integer, 0 or 1.

1 no rows will be highlighted when selected
0 all selected rows will be highlighted when selected (default)

When *DisableRowHighlight* is set to 1, no rows will be highlighted if the user selects them or if they are selected by calling the commands ***AL_SetLine*** (page 87) or ***AL_SetSelect*** (page 87). AreaList Pro will still maintain a list of the selected rows, even though they will not be highlighted. Thus the commands ***AL_GetLine*** or ***AL_GetSelect*** will still return the correct selected row(s). This parameter is especially useful if you want to have a different way of showing selected rows such as by having a column of check marks or bullets.

AL_SetRowOpts can be used in both the **Before** phase and **During** phase of a script or procedure.

Examples:

```
`setup the list for single-line selection, allow the user to
```

```
  `select no lines, don't allow the user to drag lines,
  `don't accept a drag from another AreaList Pro object,
  `don't move the row style and color info with the row
  `don't disable row highlighting
AL_SetRowOpts(eNames;0;1;0;0;0;0)

  `setup the list for multi-line selection, require one line selection,
  `allow the user to Option-drag lines only within the list,
  `accept a drag from another AreaList Pro object,
  `move the row style and color info with the row
  `disable row highlighting
AL_SetRowOpts(eList;1;0;1;1;1;1)

  `setup the list for single-line selection, require one line selection,
  `allow the user to drag lines within the list and out of the list without the
                    Option key,
  `accept a drag from another AreaList Pro object,
  `move the row style and color info with the row
  `disable row highlighting
AL_SetRowOpts(eList;0;0;6;1;1;1)
```

AL_SetColOpts

AL_SetColOpts*(AreaName;AllowColumnResize;ResizeInDuring;AllowColumnLock; HideLastColumns;DisplayPixelWidth;DragColumn;AcceptDrag)*

Parameter	**Type**	**Description**
AreaName	longint	name of AreaList Pro object on layout
AllowColumnResize	integer	user resizable columns
ResizeInDuring	integer	automatically resize columns in the During phase
AllowColumnLock	integer	allow user to lock columns
HideLastColumns	integer	number of columns from the right to hide
DisplayPixelWidth	integer	display column widths
DragColumn	integer	drag a column to this or another object
AcceptDrag	integer	accept drag from another AreaList Pro object

AL_SetColOpts is used to control several AreaList Pro options pertaining to columns.

Note: the DragColumn and AcceptDrag parameters are ignored when using the Macintosh Drag Manager routines provided in AreaList Pro v5.1 and later. Please read “Dragging Commands” on page 129 and “Obsolete Dragging Commands” on page 177 for more information.

AllowColumnResize — Integer, 1 or 0. This parameters controls whether the user can resize column by clicking on the dividing line between column headers.

1 allow the user to resize columns (default)

0 do not allow the user to resize columns

When the *HideHeaders* parameter of ***AL_SetMiscOpts*** (page 66) is set to 1 (headers are hidden), *AllowColumnResize* is set to 0 internally by AreaList Pro.

ALProEvt will be set to -3 if the user resizes a column (see “Determining the User’s Action on an AreaList Pro Object” on page 145). You can get the column widths using ***AL_GetWidths*** (page 148).

ResizeInDuring — Integer, 1 or 0.

1 Whenever an array or field command is called in the **During** phase, the columns will be resized as they are in the **Before** phase. The widths used will be the last ones passed using ***AL_SetWidths*** (page 51). If any column widths are 0, then AreaList Pro will automatically calculate the width based upon the contents of the column.
0 no columns will be resized (default)

AllowColumnLock — Integer, 1 or 0.

1 enables the column lock area of the AreaList Pro object, allowing the user to modify the number of locked columns (default)
0 disables the column lock area, which prevents the user from modifying the number of locked columns

ALProEvt will be set to -4 if the user changes the column lock position (see “Determining the User’s Action on an AreaList Pro Object” on page 145). You can determine the current column lock position using ***AL_GetColLock*** (page 153).

HideLastColumns — Integer. This parameter specifies the number of columns from the right to *not* display.

0	forces the display of all columns (default).
1 to (number of columns-1)	number of columns to hide

This parameter is used when an ID column is needed for **SEARCH** purposes after the list is displayed, but you don’t want to clutter the display with the ID values. You would pass the ID array as the last array to ***AL_SetArraysNam*** (page 41), and hide the last column using this parameter with a value of one. Any pre-sort or user-sort will include the hidden column(s), to keep the values in all the columns “lined-up.” If the number of columns passed to AreaList Pro is less than or equal to the value specified by *HideLastColumns*, then only the first column will be displayed.

DisplayPixelWidth — Integer, 1 or 0. Used during development to allow you to easily determine what pixel width looks best for each column. Which this option is enabled, a button in the lower right area of the AreaList Pro object is enabled to toggle the headers between displaying pixel widths and the actual header values. When AreaList Pro is initially displayed, the column headers are shown. Click on the button to toggle the headers to display the pixel width.

1 column headers display the width in pixels of each column, and are updated after the user resizes the column
0 turns the pixel width display off and disables the button (default)

When pixel widths are displayed in the headers of the AreaList Pro area, the cursor will change to display a pixel count when it is over the AreaList Pro area. If the cursor is moved over one of the rows in the area and clicked, the count shown in the pointer will be updated. This value is the necessary height of the AreaList Pro object to allow the row clicked on to be the bottom one displayed. This feature is disabled whenever the column widths are not displayed. *Please read the section "Column Widths" on page 30 for more information.*

DragColumn — Integer, 0, 1, 2, or 3. This parameter controls if, and how, columns may be dragged.

0 do not allow a column to be dragged (default)
1 allow a column to be dragged within, but not out of the AreaList Pro object
2 allow a column to be dragged out of, but not within the AreaList Pro object
3 allow a column to be dragged both within and out of the AreaList Pro object

If the *UserSort* option of ***AL_SetSortOpts*** is disabled, column dragging will begin immediately after the user clicks in the column header, and an outline of the column will appear. If user sorting is enabled, the drag begins when the pointer is moved 20 pixels outside of the column to the left or right, or 30 pixels above or below the header area. It is up to you to keep track of the new position of the columns: dragging the first column to the right will cause the second column to become the first. Future calls to AreaList Pro code should take these changes into account. See "***AL_GetDragCol***" on page 183 and "Dragging Commands" on page 129 for more information.

AcceptDrag — Integer, 1 or 0. This parameter controls whether columns may be dragged into the AreaList Pro object *AreaName*.

1 this AreaList Pro object will accept a column dragged from another AreaList Pro object
0 this AreaList Pro object will not accept a column (default)

AL_SetColOpts can be used in both the **Before** phase and **During** phase of a script or procedure.

Examples:

```
  `allow user to resize columns, don't resize columns in the
  `During phase, allow column lock, hide the last two columns,
  `disable the pixel width display, don't allow or accept column dragging
AL_SetColOpts(eNames;1;0;1;2;0;0;0)

  `don't allow user to resize columns, resize columns in the
  `During phase, allow column lock, don't hide any columns,
  `enable the pixel width display, don't allow column dragging, but accept
                    dragged columns
AL_SetColOpts(eNames;0;1;1;0;1;0;1)
```

AL_SetCellOpts

AL_SetCellOpts*(AreaName; CellSelection; MoveWithData; Optimization)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
CellSelection	integer	cell selection mode
MoveWithData	integer	move cell attributes with data
Optimization	integer	optimize cell attribute allocation

AL_SetCellOpts is used to set options specific to cells.

CellSelection — Integer, 0, 1, 2.

0 row selection is enabled according to the *MultiLine* option of ***AL_SetRowOpts*** (page 59) (default)
1 only one cell at a time may be selected (single cell selection)
2 many cells may be selected, contiguous or discontiguous (multiple cell selection)

Note: when CellSelection is set to a value of 1 or 2, row dragging is disabled.

MoveWithData — Integer, 0 or 1. This parameter is used with ***AL_SetCellStyle*** (page 79) and ***AL_SetCellColor*** (page 81).

1 cell attributes (not including cell selection) will move with the cell after sorting, row dragging, or column dragging (default)
0 cell attributes will not move

Optimization — Integer, 1 to 5. The default is 1.

A value of 1 means that the block used to store the cell attributes (per row) is grown a small chunk at a time. A value of 5 means that the block used to store the cell attributes is grown a large chunk at a time. Thus a lower number means that setting cell attributes may be slower but will (potentially) require less memory. Conversely, a higher number means that setting cell attributes may be faster but require more memory.

Optimization should not be set above 1 unless the number of columns in the AreaList Pro object is greater than 10 and a large percentage of the cells will have their cell attributes set.

Example:

AL_SetCellOpts(eList;1;1;1) `single cell selection only, move data with cells, normal optimization

AL_SetMiscOpts

AL_SetMiscOpts*(AreaName;HideHeaders;AreaSelected;PostKey;ShowFooters)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
HideHeaders	integer	hide the column headers
AreaSelected	integer	visual cue that external area is selected
PostKey	string	character to post to execute script
ShowFooters	integer	show the column footers

AL_SetMiscOpts is used to control several AreaList Pro options.

HideHeaders — Integer, 1 or 0.

1 the column headers will not be displayed
0 the column headers will be displayed (default)

When *HideHeaders* is 1, the *AllowColumnResize* parameter of ***AL_SetColOpts*** (page 62) is set to 0 internally by AreaList Pro.

AreaSelected — Integer, 0, 1, or 2. This parameter controls how the AreaList Pro object is displayed when it is “selected” (i.e., the active layout object).

0 no indication will be given to the user that the external area is selected (default)
1 a 2-pixel wide border will be drawn around the external area when it is selected
2 a System 7 style selection rectangle will be drawn around the external area when it is selected

PostKey — One character string. AreaList Pro causes the script of an AreaList Pro external object to run by posting a keyboard event to the Macintosh Event Queue. This parameter is used to specify what character to post. The default is the backslash character (“\”). *Please read the section “AreaList Pro’s PostKey” on page 145 for more information.*

ShowFooters — Integer, 1 or 0. This parameter controls whether footers are displayed for the AreaList Pro object *AreaName*. Footers are displayed using ***AL_SetFooters*** (page 51).

1 footers will be displayed below each column
0 footers will not be displayed (default)

See “Footers” on page 29, ***AL_SetFooters*** (page 51) and ***AL_SetFtrStyle*** (page 57) for more information about footers.

AL_SetMiscOpts can be used in both the **Before** phase and **During** phase of a script or procedure.

Examples:

```
  `don't hide the headers, show the area selected cue,
  `use the default post-key, don't show footers
AL_SetMiscOpts(eNames;0;1;"";0)

  `hide the headers, don't show the area selected cue,
  `use <cmd>open bracket for the post-key, show footers
AL_SetMiscOpts(eNames;1;0;"[";1)
```

IncludeHiddenCols	integer	include hidden columns in Edit menu copy
FieldDelimiter	string	field separator for Edit menu copy
RecordDelimiter	string	record separator for Edit menu copy
FieldWrapper	string	field wrapper for Edit menu copy

AL_SetCopyOpts

AL_SetCopyOpts*(AreaName;IncludeHiddenCols;FieldDelimiter;RecordDelimiter; FieldWrapper)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout

AL_SetCopyOpts is used to control several AreaList Pro options pertaining to copying the selected line(s) when "Copy" is selected from the Edit menu. Because of limitations of the Macintosh clipboard, picture columns cannot be copied to the clipboard; a blank field will be copied instead.

IncludeHiddenCols — Integer, 1 or 0.

1 any values in hidden columns will be included when the user uses the Edit menu Copy command
0 any values in hidden columns will *not* be included when the user uses the Edit menu Copy command (default)

FieldDelimiter — One character string. The delimiter used to separate fields when the user copies selected lines to the clipboard. Default is the TAB character (ASCII 9).

RecordDelimiter — One character string. The delimiter used to separate lines when the user copies selected lines to the clipboard. Default is the carriage return character (ASCII 13).

FieldWrapper — One character string. The character used to "wrap" fields when the user copies selected lines to the clipboard. This character will be placed both before and after each field. If *FieldWrapper* is the null string, then no character will wrap the fields. The default is that no character will wrap the fields.

FieldWrapper will be especially useful on Windows because programs such as Excel or Works expect text to be pasted in with commas separating, and quotes wrapping the fields.

AL_SetCopyOpts can be used in both the **Before** phase and **During** phase of a script or procedure.

Examples:

```
  `include hidden columns in Edit menu Copy,
  `use the default Field and Record delimiters for Edit menu Copy
AL_SetCopyOpts(eNames;1;"";"")

  `don't include hidden columns in Edit menu Copy,
  `use different Field and Record delimiters for Edit menu Copy
AL_SetCopyOpts(eNames;0;Char(241);Char(242))
```

AL_SetSortOpts

AL_SetSortOpts*(AreaName;SortInDuring;UserSort;AllowSortEditor;SortEditorPrompt; ShowSortOrder)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
SortInDuring	integer	automatically sort in the During phase
UserSort	integer	allow user to sort
AllowSortEditor	integer	allow user to sort with editor
SortEditorPrompt	string	set the prompt of the Sort Editor
ShowSortOrder	integer	show the current sort order in the Sort Editor

AL_SetSortOpts is used to control several AreaList Pro options pertaining to sorting.

SortInDuring — Integer, 1 or 0.

1 whenever an array or field command is called in the **During** phase, the columns will be automatically sorted based upon the current sort order

0 no sorting will be done automatically in the **During** phase (default)

UserSort — Integer, 0, 1, 2 or 3.

0 Disable the user sort buttons in the column headers

1 Enable the user sort buttons in the column headers (default). The sort buttons will highlight when clicked, and the columns sorted based on the values in the column which was clicked. The AreaList Pro area's script will run, with ALProEvt = - 1.

2 Bypass the user sort buttons in the column headers. The sort buttons will highlight when clicked, but no sort will be performed. The AreaList Pro area's script will run, with ALProEvt = - 1. This allows you to procedurally check for a click of a sort button by the user and perform your own sort action.

3 Enable the user sort buttons for indexed fields only. If the field in the column is not indexed, the sort button will highlight when clicked, but no sort will be performed. If the field in the column is indexed, the fields will be sorted based on the values in the column which was clicked. The AreaList Pro area's script will run, with ALProEvt = - 1. If arrays, not fields, are displayed in the object then all of the sort buttons will be enabled.

If the value of *UserSort* is 1, 2 or 3, and the column contains a picture column, the column header will highlight, but no sort will occur, and the script for the AreaList Pro area will not run.

If the value of *UserSort* is 1 or 3, and the column contains a field from a related one file, the column header will highlight, but no sort will occur, and the script for the AreaList Pro area will not run. If the value of *UserSort* is 2, and the column contains a field from a related one file, the column header will highlight, but no sort will occur, and the script for the AreaList Pro area will run.

When the user sort is bypassed by setting *UserSort* to 2, ***AL_GetSort*** (page 149) is still used to get the column header that was clicked on.

AllowSortEditor — Integer, 1 or 0.

1 the user can command-click in the header to display the AreaList Pro Sort Editor
0 the user is not able to display the Sort Editor (default)

The AreaList Pro Sort Editor can also be displayed with ***AL_ShowSortEd*** (page 157).

SortEditorPrompt — String (optional). This is the prompt that will be displayed at the top of the AreaList Pro Sort Editor. The default is “AreaList™ Pro Sort Editor”. The default prompt is stored in a DITL resource which is part of the AreaList Pro external. If you are familiar with ResEdit, you can change this default value.

ShowSortOrder — Integer, 1 or 0.

1 the current sort order will be displayed in the Sort Order list whenever the AreaList Pro Sort Editor is displayed
0 the Sort Order list will be empty whenever the AreaList Pro Sort Editor is displayed (default)

AL_SetSortOpts can be used in both the **Before** phase and **During** phase of a script or procedure.

Examples:

```
  `don't automatically sort in During phase, allow user to sort with buttons,
  `allow user to invoke Sort Editor, display the default Sort Editor prompt,
  `don't show the current sort order in the Sort Editor
AL_SetSortOpts(eNames;0;1;1;"";0)

  `automatically sort in During phase, don't allow user to sort with buttons,
  `allow user to invoke Sort Editor, change the Sort Editor prompt,
  `show the current sort order in the Sort Editor
AL_SetSortOpts(eNames;1;0;1;"People Sort Order";1)
```

AL_SetForeClr

AL_SetForeClr*(AreaName;ColumnNum;HdrForeColor1;HdrForeColor2;ListForeColor1;ListForeColor2;FtrForeColor1;FtrForeColor2)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
ColumnNum	integer	number of column
HdrForeColor1	string	header foreground color from AreaList Pro’s palette
HdrForeColor2	integer	header foreground color from 4D’s palette
ListForeColor1	string	list foreground color from AreaList Pro’s palette
ListForeColor2	integer	list foreground color from 4D’s palette
FtrForeColor1	string	footer foreground color from AreaList Pro’s palette
FtrForeColor2	integer	footer foreground color from 4D’s palette

AL_SetForeClr is used to specify the foreground color for a column header, a list area column, and a column footer.

AreaList Pro has its own palette, with the following colors: white, black, blue, green, yellow, magenta, red, cyan, gray, light gray.

ColumnNum — The column for which to set the foreground color. Use a value of zero (0) for *ColumnNum* to apply the parameters to all columns.

HdrForeColor1 — String, name of the color in AreaList Pro’s palette. This will be the foreground color for the column header. If the name is not in AreaList Pro’s palette or it is a null string, then *HdrForeColor2* will be used.

HdrForeColor2 — Integer, 1 to 256. The color at this position in 4D’s palette will be used for the foreground color for the column header.

ListForeColor1 — String, name of the color in AreaList Pro’s palette. This will be the foreground color for the column. If the name is not in AreaList Pro’s palette or it is a null string, then *ListForeColor2* will be used.

ListForeColor2 — Integer, 1 to 256. The color at this position in 4D’s palette will be used for the foreground color for the column.

FtrForeColor1 — String, name of the color in AreaList Pro’s palette. This will be the foreground color for the column footer. If the name is not in AreaList Pro’s palette or it is a null string, then *FtrForeColor2* will be used.

FtrForeColor2 — Integer, 1 to 256. The color at this position in 4D’s palette will be used for the foreground color for the column footer.

If ***AL_SetForeClr*** is not called, the default is black for the header, list, and footer foreground colors.

AL_SetForeClr can be used in both the **Before** phase and **During** phase of a script or procedure.

Examples:

```
  `red for column header foreground,
  `light gray for column foreground (all columns)
  `Blue for footer foreground
AL_SetForeClr(eNames;0;"Red";0;"Light Gray";0;"Blue";0)

  `green for column header foreground,
  `13th color from 4D's palette for column foreground (4th column)
  `7th color from 4D's palette for footer foreground
AL_SetForeClr(eNames;4;"Green";0;"";13;"";7)
```

AL_SetBackClr

AL_SetBackClr*(AreaName;HdrBackColor1;HdrBackColor2;ListBackColor1;ListBackColor2; FtrBackColor1;FtrBackColor2)*

Parameter	**Type**	**Description**
AreaName	longint	name of AreaList Pro object on layout
HdrBackColor1	string	header background color from AreaList Pro’s palette
HdrBackColor2	integer	header background color from 4D’s palette
ListBackColor1	string	list background color from AreaList Pro’s palette
ListBackColor2	integer	list background color from 4D’s palette
FtrBackColor1	string	footer background color from AreaList Pro’s palette
FtrBackColor2	integer	footer background color from 4D’s palette

AL_SetBackClr is used to specify the background color for the header, list area, and footer. While the foreground color can be specified for each column, the background color for the header, list area, or footer can only be specified for all columns.

AreaList Pro has its own palette, with the following colors: white, black, blue, green, yellow, magenta, red, cyan, gray, light gray.

HdrBackColor1 — String, name of the color in AreaList Pro’s palette. This will be the background color for the column header. If the name is not in AreaList Pro’s palette or it is a null string, then *HdrBackColor2* will be used.

HdrBackColor2 — Integer, 1 to 256. The color at this position in 4D’s palette will be used for the background color for the column header.

ListBackColor1 — String, name of the color in AreaList Pro’s palette. This will be the background color for the column. If the name is not in AreaList Pro’s palette or it is a null string, then *ListBackColor2* will be used.

ListBackColor2 — Integer, 1 to 256. The color at this position in 4D’s palette will be used for the background color for the column.

FtrBackColor1 — String, name of the color in AreaList Pro’s palette. This will be the background color for the footer. If the name is not in AreaList Pro’s palette or it is a null string, then *FtrBackColor2* will be used.

FtrBackColor2 — Integer, 1 to 256. The color at this position in 4D’s palette will be used for the background color for the footer.

If ***AL_SetBackClr*** is not called, the default is white for the header, list, and footer background colors.

AL_SetBackClr can be used in both the **Before** phase and **During** phase of a script or procedure.

Examples:

```
  `light gray for header background,
  `white for list background
  `gray for the footer background
AL_SetBackClr(eNames;"Light Gray";0;"White";0;"Gray";0)

  `white for header background,
  `13th color from 4D's palette for list background
  `Color 246 from 4D's palette for footer background
AL_SetBackClr(eNames;"White";0;"";13;"";246)
```

AL_SetDividers

AL_SetDividers*(AreaName;ColDividerPattern;ColDividerColor1;ColDividerColor2;RowDividerPattern;RowDividerColor1;RowDividerColor2)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
ColDividerPattern	string	pattern of the column divider
ColDividerColor1	string	color from AreaList Pro’s palette for the column divider
ColDividerColor2	integer	color from 4D’s palette for the column divider
RowDividerPattern	string	pattern of the row divider
RowDividerColor1	string	color from AreaList Pro’s palette for the row divider
RowDividerColor2	integer	color from 4D’s palette for the row divider

AL_SetDividers is used to set the pattern and color of the column and row dividers.

These are the available patterns: white, black, gray, light gray, and dark gray.

AreaList Pro has its own palette, with the following colors: white, black, blue, green, yellow, magenta, red, cyan, gray, light gray.

ColDividerPattern — String, name of the pattern for the column divider. If a null string is used then no column divider will be displayed. Do not use a Light Gray or Dark Gray pattern for the column divider. For technical reasons, this will cause the column divider to change appearance as the list is scrolled.

ColDividerColor1 — String, name of the color in AreaList Pro’s palette. This will be the color for the column divider. If the name is not in AreaList Pro’s palette or it is a null string, then *ColDividerColor2* will be used.

ColDividerColor2 — Integer, 1 to 256. The color at this position in 4D’s palette will be used for the column divider.

RowDividerPattern — String, name of the pattern for the row divider. If a null string is used then no row divider will be displayed.

RowDividerColor1 — String, name of the color in AreaList Pro’s palette. This will be the color for the row divider. If the name is not in AreaList Pro’s palette or it is a null string, then *RowDividerColor2* will be used.

RowDividerColor2 — Integer, 1 to 256. The color at this position in 4D’s palette will be used for the row divider.

If ***AL_SetDividers*** is not called, then no column or row dividers will be displayed.

AL_SetDividers can be used in both the **Before** phase and **During** phase of a script or procedure.

Examples:

```
  `display solid gray column dividers and no row dividers
AL_SetDividers(eNames;"Black";"Gray";0;"";"";0)

  `display column and row dividers in a gray pattern
AL_SetDividers(eNames;"Gray";"Black";0;"Gray";"Black";0)
```

AL_SetRowStyle

AL_SetRowStyle*(AreaName;RowNum;StyleNum;FontName)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
RowNum	integer	number of row
StyleNum	integer	style of the font
FontName	string	name of the font

AL_SetRowStyle is used to set the type style and font for a particular row. It will override the style and font settings for all columns in that row. The size settings of each column will still apply.

RowNum — Integer. The row for which to set the style. Use a value of zero (0) for *RowNum* to apply the parameters to all rows.

StyleNum — Integer. This parameter is used to set the style for the row. The different values in the table below can be added together to produce combinations of styles. For example, bold italic has a value of 3.

Style	Number
Plain	0
Bold	1
Italic	2
Underline	4
Outline	8
Shadow	16
Condensed	32
Extended	64

If a row style has been previously set, it may be removed by setting *StyleNum* to -1. This may also be applied to all rows by passing a zero (0) for the row number. This will have no effect on rows that have not been previously set.

The row style may be left unchanged by setting *StyleNum* to 256.

FontName — String. This parameter specifies the font for a row. If a row font has been previously set, it may be removed by setting *FontName* to “-1”. *Note that the value is a string, not a number.* This may also be applied to all rows by passing a zero (0) for the row number. This will have no effect on rows that have not been previously set.

The row font may be left unchanged by setting *FontName* to the empty string ("").

See the *MoveWithData* option of ***AL_SetRowOpts*** (page 59). This controls whether row styles stay with their rows whenever sorting or dragging occurs.

Examples:

AL_SetRowStyle(eNames;10;2;"") `Set row 10 to be italic
AL_SetRowStyle(eNames;0;1;"Helvetica") `Set all rows to be bold, Helvetica
AL_SetRowStyle(eNames;0;-1;"-1") `Reset all row styles. Column settings will be used

```
  `set the 12th row to display the Times font in bold italic style
AL_SetRowStyle(eList;12;3;"Times")
If(During)
  AL_UpdateArrays(eList;-1)
End if
```

AL_SetRowColor

AL_SetRowColor*(AreaName;RowNum;RowForeColor1;RowForeColor2; RowBackColor1; RowBackColor2)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
RowNum	integer	number of row
RowForeColor1	string	row foreground color from AreaList Pro's palette
RowForeColor2	integer	row foreground color from 4D's palette
RowBackColor1	string	row background color from AreaList Pro's palette
RowBackColor2	integer	row background color from 4D's palette

AL_SetRowColor is used to specify the foreground and background color for a row. It will override the foreground and background color settings for all columns in that row.

AreaList Pro has its own palette, with the following colors: white, black, blue, green, yellow, magenta, red, cyan, gray, light gray.

RowNum — Integer. The row for which to set the foreground color. Use a value of zero (0) for *RowNum* to apply the parameters to all rows.

RowForeColor1 — String. Name of the color in AreaList Pro's palette. This will be the foreground color for the row. If the name is not in AreaList Pro's palette or it is a null string, then *RowForeColor2* will be used.

RowForeColor2 — Integer, 1 to 256. Foreground color number for the row (from 4D's palette). If a row color has been previously set, it may be removed by setting *RowForeColor1* to an empty string (“”), and *RowForeColor2* to -1. This may also be applied to all rows by passing a zero (0) for the *RowNum* . This will have no effect on rows that have not been previously set.

The row foreground color may be left unchanged by setting *RowForeColor1* to the empty string (""), and *RowForeColor2* to 0.

RowBackColor1 — String. Name of the color in AreaList Pro's palette. This will be the background color for the row. If the name is not in AreaList Pro’s palette or it is the empty string (""), then *RowBackColor2* will be used.

RowBackColor2 — Integer, 1 to 256. Background color number for the cell (from 4D’s palette). If a row background color has been previously set, it may be removed by setting *RowBackColor1* to the empty string (""), and *RowBackColor2* to -1. This may also be applied to all rows by passing a zero (0) for the row number. This will have no effect on rows that have not been previously set.

The row background color may be left unchanged by setting *RowBackColor1* to the empty string (""), and *RowBackColor2* to 0.

See the *MoveWithData* option of ***AL_SetRowOpts*** (page 59). This controls whether row colors stay with their rows whenever sorting or dragging occurs.

Examples:

```
AL_SetRowColor(eNames;10;"Blue";0;"Light gray";0) `Set row 10 to foreground blue, background light gray
AL_SetRowColor(eNames;0;"Blue";0;"Yellow";0)   `Set all rows to blue foreground, yellow background
AL_SetRowColor(eNames;0;"";-1;"";-1)   `Reset all row colors to use the column color settings
  `set the 10th row to display a foreground color of Blue and background color of Light gray
AL_SetRowColor(eList;10;"Blue";0;"Light Gray";0)
  `set the 12th row to display a foreground color of Green and the current background color
AL_SetRowColor(eList;12;"Green";0;"";0)
If(During)
  AL_UpdateArrays(eList;-1)
End if
```

AL_SetCellStyle

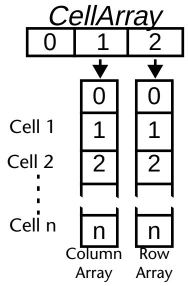
AL_SetCellStyle*(AreaName; Cell1Col; Cell1Row; Cell2Col; Cell2Row; CellArray; StyleNum; FontName)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
Cell1Col	integer	first cell column
Cell1Row	integer	first cell row
Cell2Col	integer	last cell column
Cell2Row	integer	last cell row
CellArray	array	discontiguous cells
StyleNum	integer	style of the font
FontName	string	name of the font

AL_SetCellStyle is used to set the font and/or style of a specific cell, range of cells, or list of cells.

- **To specify a single cell.** If *Cell1Col* and *Cell1Row* are greater than 0 and *Cell2Col* or *Cell2Row* are less than or equal to 0 then only [*Cell1Col*, *Cell1Row*] will be set.
- **To specify a range of cells.** If *Cell1Col* and *Cell1Row* are greater than 0 and *Cell2Col* and *Cell2Row* are greater than 0 then the range of cells from [*Cell1Col*, *Cell1Row*] to [*Cell2Col*, *Cell2Row*] will be set.
- **To specify discontiguous cells.** If *Cell1Col* or *Cell1Row* are less than or equal to 0 then the cells in *CellArray* will be set.

CellArray — two-dimensional integer array. The first dimension must be two. The first array is for the column indices and the second array is for the row indices. The second dimension must be the same as the number of cells that are to be selected. See the following illustration.



StyleNum — Integer. This parameter is used to set the style for the specified cells. The values shown below can be added together to combine styles.

Style	Number
Plain	0
Bold	1
Italic	2
Underline	4
Outline	8
Shadow	16
Condensed	32
Extended	64

If a cell style has been previously set, the style may be removed by setting *StyleNum* to -1. The cell style may be left unchanged by setting *StyleNum* to 256.

FontName — String. If a cell font has been previously set, it may be removed by setting *FontName* to “-1”. Note that the value is a string, not a number. The cell font may be left unchanged by setting *FontName* to the empty string ("").

See the *MoveWithData* option of ***AL_SetCellOpts*** (page 65). This controls whether cell styles and fonts stay with their cells whenever sorting, row dragging, or column dragging occurs.

Example:

```
  `set the currently highlighted cell(s) to be bold
ARRAY INTEGER(aInt;2;0)
$Result:=AL_GetCellSel(eList;vCol1;vRow1;vCol2;vRow2;aInt)
If($Result=1)
  AL_SetCellStyle(eList;vCol1;vRow1;0;0;aInt;1;"")
  If(During)
    AL_UpdateArrays(eList;-1)
  End if
End if
```

AL_SetCellColor

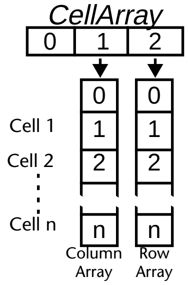
AL_SetCellColor*(AreaName;Cell1Col;Cell1Row;Cell2Col;Cell2Row;CellArray; ForeColor1; ForeColor2;BackColor1;BackColor2)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
Cell1Col	integer	first cell column
Cell1Row	integer	first cell row
Cell2Col	integer	last cell column
Cell2Row	integer	last cell row
CellArray	array	discontiguous cells
ForeColor1	string	foreground color from AreaList Pro's palette
ForeColor2	integer	foreground color from 4D's palette
BackColor1	string	background color from AreaList Pro's palette
BackColor2	integer	background color from 4D's palette

AL_SetCellColor is used to set the foreground color and/or background color of a specific cell, range of cells, or list of cells.

- **To specify a single cell.** If *Cell1Col* and *Cell1Row* are greater than 0 and *Cell2Col* or *Cell2Row* are less than or equal to 0 then only [*Cell1Col*, *Cell1Row*] will be set.
- **To specify a range of cells.** If *Cell1Col* and *Cell1Row* are greater than 0 and *Cell2Col* and *Cell2Row* are greater than 0 then the range of cells from [*Cell1Col*, *Cell1Row*] to [*Cell2Col*, *Cell2Row*] will be set.
- **To specify discontiguous cells.** If *Cell1Col* or *Cell1Row* are less than or equal to 0 then the cells in *CellArray* will be set.

CellArray — two-dimensional integer array. The first dimension must be two. The first array is for the column indices and the second array is for the row indices. The second dimension must be the same as the number of cells that are to be selected. See the following illustration.



ForeColor1 — String. Name of the color in AreaList Pro's palette.

This will be the foreground color for the cell. If the name is not in AreaList Pro's palette or it is the empty string (""), then ForeColor2 will be used.

ForeColor2 — Integer, 1 to 256. Foreground color number for the cell (from 4D's palette). If a cell foreground color has been previously set, it may be removed by setting *ForeColor1* to the empty string (""), and *ForeColor2* to -1. The cell foreground color may be left unchanged by setting *ForeColor1* to the empty string (""), and *ForeColor2* to 0.

BackColor1 — String. Name of the color in AreaList Pro's palette. This will be the background color for the cell. If the name is not in AreaList Pro's palette or it is the empty string (""), then *BackColor2* will be used.

BackColor2 — Integer, 1 to 256. Background color number for the cell (from 4D's palette). If a cell background color has been previously set, it may be removed by setting *BackColor1* to the empty string (""), and *BackColor2* to -1. The cell background color may be left unchanged by setting *BackColor1* to the empty string (""), and *BackColor2* to 0.

The foreground and background colors for a cell may be set differently during data entry by calling ***AL_SetCellColor*** in the *entry started* procedure and again in the *entry finished* procedure to restore the colors.

See the *MoveWithData* option of ***AL_SetCellOpts*** (page 65). This controls whether cell foreground and background colors stay with their cells whenever sorting, row dragging, or column dragging occurs.

Example:

```
  `set all negative values in the third column, a real array, to have a foreground
                  color of red
ARRAY INTEGER(aInt;2;0)  `MUST initialize a two-dimensional integer array
For($i;1;Size of array(aRevenue))  `check each element in the array
  If(aRevenue{$i}<0)  `is the value in this element negative?
    AL_SetCellColor(eList;3;$i;0;0;aInt;"Red";0;"";0)  `if so, then show it in
                  Red
  End if
End for
If(During)
  AL_UpdateArrays(eList;-1)
End if
```

AL_GetCellStyle

AL_GetCellStyle*(AreaName; CellCol; CellRow; StyleNum; FontName)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
CellCol	integer	cell column
CellRow	integer	cell row
StyleNum	integer	style of the font
FontName	string	name of the font

AL_GetCellStyle is used to get the font and/or style of a particular cell. It will not get the column or row font and/or style.

StyleNum — Integer. This parameter returns the style number for the cell. The number can be a sum of several individual styles. For example, bold italic has a value of 3.

Style	Number
Plain	0
Bold	1
Italic	2
Underline	4
Outline	8
Shadow	16
Condensed	32
Extended	64

If a cell style has not been previously set, the value of *StyleNum* will be -1.

FontName — String. If a cell font has not been previously set, the value of *FontName* will be “-1”. *Note that the value is a string, not a number.*

Example:

```
  `get the style of the cell in the third column, first row
AL_GetCellStyle(eList;3;1;vStyle;vFont)
```

AL_SetCellSel

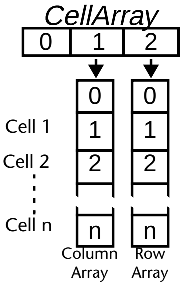
AL_SetCellSel*(AreaName; Cell1Col; Cell1Row; Cell2Col; Cell2Row; CellArray)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
Cell1Col	integer	first cell column
Cell1Row	integer	first cell row
Cell2Col	integer	last cell column
Cell2Row	integer	last cell row
CellArray	array	discontiguous cells

AL_SetCellSel is used to set the cell selection. Use the *CellSelection* option of ***AL_SetCellOpts*** (page 65) to specify a cell selection mode prior to using this command.

- **To select a single cell.** If *Cell1Col* and *Cell1Row* are greater than 0 and *Cell2Col* or *Cell2Row* are less than or equal to 0 then only [*Cell1Col*, *Cell1Row*] will be selected.
- **To select a range of cells.** If *Cell1Col* and *Cell1Row* are greater than 0 and *Cell2Col* and *Cell2Row* are greater than 0 then the range of cells from [*Cell1Col*, *Cell1Row*] to [*Cell2Col*, *Cell2Row*] will be selected.
- **To select discontiguous cells.** If *Cell1Col* or *Cell1Row* are less than or equal to 0 then the cells in *CellArray* will be selected.

CellArray — two-dimensional integer array. The first dimension must be two. The first array is for the column indices and the second array is for the row indices. The second dimension must be the same as the number of cells that are to be selected. See the following illustration.



Examples:

AL_SetCellSel (eArea;1;3;0;0) `select cell at column 1, row 3

AL_SetCellSel (eArea;2;2;5;5) `select cells from column 2, row 2 to column 5, row 5

```
ARRAY INTEGER(aCellSelect;2;4)
aCellSelect{1}{1}:=1  `column 1
aCellSelect{2}{1}:=1  `row 1
aCellSelect{1}{2}:=1  `column 1
aCellSelect{2}{2}:=2  `row 2
aCellSelect{1}{3}:=2  `column 2
aCellSelect{2}{3}:=5  `row 5
aCellSelect{1}{4}:=2  `column 2
aCellSelect{2}{4}:=6  `row 6
AL_SetCellSel (eArea;0;0;0;0;aCellSelect)   `select the cells in aCellSelect
```

AL_GetCellColor

AL_GetCellColor*(AreaName;CellCol;CellRow;ForeColor2;BackColor2)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
CellCol	integer	cell column
CellRow	integer	cell row
ForeColor2	integer	foreground color from 4D’s palette
BackColor2	integer	background color from 4D’s palette

AL_GetCellColor is used to get the foreground color and/or background color of a specific cell, range of cells, or list of cells. It will not get the column or row foreground color and/or background color.

For this command to function correctly the cell foreground and background colors must have been set from 4D’s palette. In other words, the *ForeColor2* and *BackColor2* parameters must have been used in the command ***AL_SetCellColor*** (page 81).

ForeColor2 — Integer, 1 to 256. Foreground color number of the cell (from 4D’s palette). If a cell foreground color has not been previously set, the value of *ForeColor2* will be -1.

BackColor2 — Integer, 1 to 256. Background color number of the cell (from 4D’s palette). If a cell background color has not been previously set, the value of *BackColor2* will be -1.

AL_SetSort

AL_SetSort*(AreaName;Column1; … ;ColumnN)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
Column	integer	column to perform sort upon

AL_SetSort is used to perform a multi-level sort.

Column — Integer. These parameters specify the columns to use for the sort criteria.

A *Column* greater than 0 causes an ascending sort to be performed upon that column, while a *Column* less than 0 causes a descending sort to be performed upon that column. If a *Column* is 0, or it is a picture array or field, or it contains a field from a related one file, then all subsequent columns will be ignored.

If the first *Column* has a value other than 0, then its header will be underlined. If the first *Column* has a value of 0, then AreaList Pro will not sort the columns and no header will be underlined.

If the first two *Columns* have the same value, then AreaList Pro will not sort the columns, but will underline the header for the first *Column*.

You can determine what columns a user has sorted using ***AL_GetSort*** (page 149).

Examples:

```
AL_SetSort(eNames;3;4;7)  `sort on columns 3, 4, and 7 (all ascending)
AL_SetSort(eContacts;-1;3;-2)  `sort on columns 1 (descending), 3 (ascending),
                                `and 2 (descending)
AL_SetSort(eList;0)  `don't sort, and don't underline any header
AL_SetSort(eNames;2;2)   `don't sort, but do underline the header for column 2
```

AL_SetLine

AL_SetLine*(AreaName;LineNum)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
LineNum	integer	line number to select (highlight)

AL_SetLine is used to set the line to be highlighted. This command is used in the **Before** phase to set up the initial display of an AreaList Pro object. You can also use it in the **During** phase to control what element is selected. If this command is not used, then AreaList Pro will display the columns with the first line selected.

AL_SetLine should only be used with an AreaList Pro object in single-line mode. If *AreaName* is in multi-line mode, you must use ***AL_SetSelect*** (page 87).

LineNum — Integer. This parameter specifies what line to highlight.

AL_SetLine can be used in both the **Before** phase and **During** phase of a script or procedure.

Example:

```
Case of
  :(Before)
    $Error:=AL_SetArraysNam(eList;1;3;"aFN";"aLN";"aComp")
    AL_SetLine(eList;3)   `highlight 3rd line
End case
```

AL_SetSelect

AL_SetSelect*(AreaName;RowsToSelect)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
RowsToSelect	integer array	contains element numbers to select (highlight) when the Multi-Line option is enabled

AL_SetSelect is used to set the lines to be highlighted. This command is used in the **Before** phase to setup the initial display of an AreaList Pro object. You can also use it in the **During** phase to control what elements are selected. If this command is not used, then AreaList Pro will display the columns with no lines selected.

AL_SetSelect should only be used with an AreaList Pro object in multi-line mode. If *AreaName* is in single-line mode, you must use ***AL_SetLine*** (page 87).

RowsToSelect— Integer array. This parameter contains a list of rows which you wish to select, or highlight.

AL_SetSelect can be used in both the **Before** phase and **During** phase of a script or procedure.

Example:

```
  `eNames AreaList Pro object script
Case of
  :(Before)
    ARRAY INTEGER(aLines;2)  `create an integer array with 2 elements
    aLines{1}:=1  `set line 1 to be highlighted
    aLines{2}:=3  `and line 3 to be highlighted
    $Error:=AL_SetArraysNam(eNames;1;2;"aFN";"aLN")  `specify arrays to
                    display
    AL_SetSelect(eNames;aLines)  `specify the lines to highlight
End case
```

AL_SetScroll

AL_SetScroll*(AreaName;VertScroll;HorizScroll)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
VertScroll	integer	vertical position (element #) to scroll to
HorizScroll	integer	horizontal position (column #) to scroll to

AL_SetScroll is used to set the position of the thumb in the vertical and horizontal scroll bars.

VertScroll — Integer. This parameter represents the element number to display at the top of the AreaList Pro display.

HorizScroll — Integer. This parameter represents the column number to display at the left of the AreaList Pro display.

The value passed to *HorizScroll* represents the actual column number, including any columns which might be currently locked. For example, if the two left columns are locked, and you want to scroll the list one column to the left, so that the fourth column is adjacent to the 2nd locked column, then the value to pass is four.

AL_SetScroll can also be used to hide or show the vertical and the horizontal scroll bar. The possible values to use to hide or show the scroll bars are shown in the table below. The default is that both scroll bars are shown.

Value	VertScroll	HorizScroll
>0	Vertical scroll position	Horizontal scroll position
0	Hide when changing pages (required)	Hide when changing pages (required)
-1	Hide if shown, Show if hidden	Hide if shown, Show if hidden
-2	Show	Show
-3	Hide	Hide

When using ***AL_SetScroll*** to hide or show the scroll bars, either ***AL_UpdateArrays*** (page 46) with *UpdateMethod* set to -2, or ***AL_UpdateFields*** (page 101) with *UpdateMethod* set to 2 must be called.

AL_SetScroll can still be used to set the scroll position even with the scroll bar(s) hidden.

AreaList Pro automatically hides the horizontal scroll bar if *AllowColumnResize* in ***AL_SetColOpts*** (page 62) is set to 0 and all of the displayed columns fit within the width of the list area. AreaList Pro automatically shows the horizontal scroll bar if *AllowColumnResize* in ***AL_SetColOpts*** (page 62) is set to 1 or all of the displayed columns do not fit within the width of the list area. If the horizontal scroll bar is shown or hidden manually by passing -1, -2 or -3 in the *HorizScroll* parameter of ***AL_SetScroll***, then this behavior will be permanently disabled for the AreaList Pro object.

Note: Pass values of zero forVertScroll and HorizScroll if the layout page is changing to a page that doesn't contain the AreaList Pro object. This is required to avoid interaction problems with 4D, as 4D doesn't notify an AreaList Pro object that it isn't on the current layout page.

AL_SetScroll can be used in both the **Before** phase and **During** phase of a script or procedure.

Examples:

```
  `set an AreaList Pro object to display the 15th element
  `the object is named eList
AL_SetScroll(eList;15;1)

  `Configure the AreaList Pro object not to display the vertical scroll bar
If(Before)
  `do any desired setup, then hide the vertical scroll bar
  AL_SetScroll(eList;-1;1)
End if
```

AL_SetColLock

AL_SetColLock*(AreaName;Columns)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
Columns	integer	number of columns to lock

AL_SetColLock is used to set the number of columns to lock. AreaList Pro will not allow more columns to be locked than the number of displayed columns minus two.

Columns — Integer. This parameter is used to specify the number of columns to lock.

AL_SetColLock can be used in both the **Before** phase and **During** phase of a script or procedure.

Example:

```
AL_SetColLock(eList;2)  `lock the first two columns
```

AL_SetHeight

AL_SetHeight*(AreaName;NumHeaderLines;HeaderHeightPad;NumRowLines;*
RowHeightPad;NumFooterLines;FooterHeightPad)

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
NumHeaderLines	integer	number of text lines in the header
HeaderHeightPad	integer	extra height for the header
NumRowLines	integer	number of text lines in each row
RowHeightPad	integer	extra height for each row
NumFooterLines	integer	number of text lines in the footer
FooterHeightPad	integer	extra height for the footer

AL_SetHeight is used to set the number of lines of text along with additional height padding in the header, in the rows, and in the footer. Only text and string columns can wrap to more than one line. If *NumRowLines* is set to 2 or more, text and string elements will be able to wrap into the number of lines specified for each row. Note that all rows will be given the same number of lines regardless of the actual number of lines used by a specific text or string element.

Additional padding may be set using *RowHeightPad* to allow more space between rows. Text will be centered vertically in the header or row. Note that the padding applies to the entire row and not on a line by line basis within the row.

NumHeaderLines — Integer. The number of lines in the header. Default is 1.

HeaderHeightPad — Integer. The extra height, in pixels, to give to the header. Default is 2.

NumRowLines — Integer. The number of lines to give to each row. Default is 1.

RowHeightPad — Integer. The extra height, in pixels, to give to each row. Default is 0.

NumFooterLines — Integer. The number of lines to give to the footer. Default is 1.

FooterHeightPad — Integer. The extra height, in pixels, to give to the footer. Default is 2.

Examples:

AL_SetHeight(eList;1;4;1;2;1;4) `Pad the header by 4 pixels, the rows by 2, the footers by 4

AL_SetHeight(eList;2;5;2;0;2;0) `Set header lines to 2, pad to 5 pixels, set row lines to 2, no padding, set footer lines to 2, no padding

AL_SetWinLimits

AL_SetWinLimits *(AreaName; EnableResize; MinWidth; MinHeight; MaxWidth; MaxHeight)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
EnableResize	integer	enable AreaList Pro object and its window to be resized
MinWidth	integer	minimum width of the window
MinHeight	integer	minimum height of the window
MaxWidth	integer	maximum width of the window
MaxHeight	integer	maximum height of the window

AL_SetWinLimits is used to enable resizing of the AreaList Pro object and its window, and to set the window's minimum and maximum limits.

EnableResize— Integer, 0 or 1. A value of 1 enables resizing of the AreaList Pro object and its window. A value of 0 disables resizing of the AreaList Pro object and its window.

MinWidth — Integer. This is the minimum width of the window containing the AreaList Pro object. AreaList Pro will not allow the window to be sized smaller than this width. If *MinWidth* will cause the AreaList Pro object to be less than 100 pixels wide, then *MinWidth* will be changed so that the object is exactly 100 pixels wide.

MinHeight — Integer. This is the minimum height of the window containing the AreaList Pro object. AreaList Pro will not allow the window to be sized smaller than this height. If *MinHeight* will cause the AreaList Pro object to be less than 100 pixels high, then *MinHeight* will be changed so that the object is exactly 100 pixels high.

MaxWidth — Integer. This is the maximum width of the window containing the AreaList Pro object. AreaList Pro will not allow the window to be sized larger than this width. *MaxWidth* must be greater than or equal to *MinWidth* and it must be less than or equal to the right edge of the AreaList Pro object as it is drawn on the layout.

MaxHeight — Integer. This is the maximum height of the window containing the AreaList Pro object. AreaList Pro will not allow the window to be sized larger than this height. *MaxHeight* must be greater than or equal to *MinHeight* and it must be less than or equal to the bottom edge of the AreaList Pro object as it is drawn on the layout.

Note: The minimum and maximum width and height apply to the window and not to the AreaList Pro object itself.

Example:

```
  `enable resizing for eList and its window
  `set the min width to 200 pixels and the min height to 150 pixels
  `set the max width to 600 pixels and the max height to 400 pixels
AL_SetWinLimits (eList;1;200;150;600;400)
```

AL_DoWinResize

AL_DoWinResize *(AreaName)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout

AL_DoWinResize is used to resize the *AreaName* object and its window. This command must be called when *ALProEvt* is equal to -9 (the AreaList Pro object was resized).

Example:

```
Case of
  : (During)

    Case of
     : (ALProEvt=-9)   `eList was resized
        AL_DoWinResize (eList)   ` Resize eList and its window

     : (ALProEvt=1)   `single click
        `Do something here
    End case   `ALProEvt
End case   `During
```

Field and Record Commands

AreaList Pro uses the new **SubselectionToArray** command in 4D to get the records for display. This command is available beginning with 4D v3.5.3. Therefore fields can not be displayed in an AreaList Pro object when used with an earlier version of 4D.

Up to 255 fields (columns) can be displayed in an AreaList Pro object.

Using the Field Display Capability

Temporary Arrays

AreaList Pro internally uses interprocess 4D arrays to get the record data from 4th Dimension. These arrays must be declared in 4D. A text file has been included that contains these declarations. Simply create a 4D global procedure named *Compiler_ALP* and copy these declarations into it. There is no need to call this procedure from your 4D code, AreaList Pro will call it for you. This procedure must exist whether your database is interpreted or compiled.

Do not access the data within these temporary arrays. These arrays are for AreaList Pro's internal use only and their contents may change at any time.

Only 30 arrays of each of the 9 data types that AreaList Pro supports are declared. If you will be displaying more than 30 fields of a certain type, then you must add more declarations within the *Compiler_ALP* global procedure. Conversely, you may remove some of these declarations if you never display fields (or display very few fields) of a certain type. Be very careful (when adding or removing declarations) to follow exactly the syntax of the existing declarations.

Arrays and Fields

To change the display from arrays to fields, first call ***AL_RemoveArrays*** (page 45) to remove all of the arrays before calling any field commands. To change the display from fields to arrays, first call ***AL_RemoveFields*** (page 101) to remove all of the fields before calling any array commands.

Note: Arrays and fields may not be displayed together in the same AreaList Pro object. If arrays are displayed in an object, then the field commands will be ignored. Conversely, if fields are displayed in an object, then the array commands will be ignored.

Fields from a Related One File

Fields from a main file and from related one files may be displayed in the same AreaList Pro object. See the commands ***AL_SetFile*** (page 98) and ***AL_SetFields*** (page 99) for further information about displaying fields from related one files.

Redraw and Scrolling

When 4D fields are displayed, the visible rows are cached (held in memory). This is done to improve redraw speed. Every field within the visible rows are held in memory so horizontal scrolling is as fast as when displaying arrays. Vertical scrolling will be slower since the records not in view have to be retrieved from 4D.

TypeAhead

Keyboard typeahead will be disabled when displaying fields.

Copy rows to the clipboard

Copying rows to the clipboard will not be allowed when displaying fields. The “Copy” menu item will be disabled when fields are displayed.

Enterability

Columns containing fields from a related one file will not be enterable either by typing or by using popups.

Dragging

When displaying arrays, AreaList Pro will rearrange the rows automatically when the user drags a row within the list. When displaying fields, AreaList Pro will not rearrange the rows automatically when the user drags a row within the list. Thus the *MoveWithData* option of ***AL_SetRowOpts*** (page 59) and the *MoveWithData* option of ***AL_SetCellOpts*** (page 65) do not apply when fields are displayed and the user drags a row within the list.

Sorting

- Indexed fields will be bold in the Sort Editor.
- Fields from related one files will be dimmed in the Sort Editor.
- Columns containing fields from a related one file will not be sorted when their column header is clicked upon.

When fields are displayed the *MoveWithData* option of ***AL_SetRowOpts*** (page 59) will be ignored when sorting. The row style and color information will not move with the row when the AreaList Pro object is sorted.

When fields are displayed the *MoveWithData* option of ***AL_SetCellOpts*** (page 65) will be ignored when sorting. The cell style and color information will not move with the cell when the AreaList Pro object is sorted.

The *UserSort* option of ***AL_SetSortOpts*** (page 69) now includes a new selector. When *UserSort* is set to 3 and fields are being displayed, only columns containing indexed fields may be sorted by clicking on their column header.

Note: AreaList Pro uses 4th Dimension's sorting routines when sorting fields. 4D only uses indices when performing a single level sort. Indices are ignored when performing a multiple level sort. Therefore, when fields are being displayed, it would be a good idea to restrict access to the AreaList Pro Sort Editor when the selection contains more than about a thousand records.

Maximum Number of Records Displayed

AreaList Pro v6 supports a maximum of 32,750 records displayed in an AreaList Pro object. You can display a selection with a greater number of records, using ***AL_SetSubSelect*** (page 102) to specify what record range within the current selection you wish to display.

This 32,750 record limit will be removed in a future version of AreaList Pro.

Performance Issues When Displaying Fields

When AreaList Pro display fields, the automatic column sizing algorithm uses only the first 20 records (or less, if the selection contains less than 20 records) in the selection. These records are always read regardless of whether the columns are automatically or manually sized. Therefore there is no performance penalty using the automatic column sizing algorithm when displaying fields. See “Performance Issues with the Formatting Commands” on page 40 for more information.

Commands

AL_SetFile

AL_SetFile *(AreaName; FileNum) ➞ ErrorCode*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
FileNum	integer	number of 4D file
ErrorCode	integer	error code

AL_SetFile tells AreaList Pro what file is the main file from which to display records.

This command is only necessary if the field to be displayed in column one is not from the main file, but from a related one file. ***AL_SetFile*** must be called before any fields have been set, otherwise it will be ignored. If this command is not called, then AreaList Pro will use the file of the field displayed in column one as the main file.

ErrorCode — Integer. The possible values are:

0	No error	n/a
1	Not a file	check to make sure that the file represented by FileNum does exist

Example:

$Error:=***AL_SetFile*** (eList;File(»[People]))

AL_SetFields

AL_SetFields *(AreaName; FileNum; ColumnNum; NumFields; FieldNum1; … ; FieldNumN) ---> ErrorCode*

Parameter	**Type**	**Description**
AreaName	longint	name of AreaList Pro object on layout
FileNum	integer	number of 4D file
ColumnNum	integer	column at which to set the first field
NumFields	integer	number of fields to set (up to 15)
FieldNum	integer	number of 4D field
ErrorCode	integer	error code

AL_SetFields tells AreaList Pro what fields to display. Up to fifteen fields can be set at a time. Any 4D field type can be used except sub-files.

Fields from related one files may also be displayed (See "***AL_SetFile***" on page 98). A separate call to ***AL_SetFields*** must be made to set these fields. To display a related one field, pass the file number of the related one file in the FileNum parameter.

ErrorCode — Integer. The possible values are:

Value	**Error Code**	**Action**
0	No error	n/a
1	Not a file	check to make sure that the file represented by FileNum does exist
2	Not a field	check to make sure that the field represented by FieldNum does exist
3	Wrong type of field	sub-files are not allowed
4	Maximum number of fields exceeded	255 fields is the maximum
5	Wrong 4D version	The 4D version must be v3.5.3 or greater to display fields
6	Not enough memory	Increase 4D's RAM partition

Examples:

```
  `set up the eList AreaList Pro object with 5 fields, all from the same file
$Error:=AL_SetFields (eList;File(»[People]);1;5;Field(»[People]First Name);
  Field(»[People]Last Name); Field(»[People]Salary); Field(»[People]Arrival);
  Field(»[People]Male))

  `set up the eList AreaList Pro object with 4 fields, the third one from a related
                  file
$Error:=AL_SetFields (eList;File(»[People]);1;2;Field(»[People]First Name);
  Field(»[People]Last Name))
$Error:=AL_SetFields (eList;File(»[Companies]);3;1;Field(»[Compa-
                nies]Company Name))
$Error:=AL_SetFields (eList;File(»[People]);4;1;Field(»[People]Salary))

  `set up the eList AreaList Pro object with 4 fields, the first one from a related
                file
  `set the main file since the field to be set in column one is not from the main
                file,
  `but from a related one file
$Error:=AL_SetFile (eList;File(»[People]))
$Error:=AL_SetFields (eList;File(»[Companies]);1;1;Field(»[Compa-
                nies]Company Name))
$Error:=AL_SetFields (eList;File(»[People]);2;3;Field(»[People]First Name);
  Field(»[People]Last Name); Field(»[People]Salary))
```

AL_InsertFields

AL_InsertFields *(AreaName; FileNum; ColumnNum; NumFields; FieldNum1; … ; FieldNumN) ➞ ErrorCode*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
FileNum	integer	number of 4D file
ColumnNum	integer	column at which to set the first field
NumFields	integer	number of fields to set (up to 15)
FieldNum	integer	number of 4D field
ErrorCode	integer	error code

AL_InsertFields functions the same as ***AL_SetFields*** (page 99), except that the fields are inserted before *ColumnNum*.

All subsequent columns will maintain their settings. In other words, any header text, column styles, etc. will stay with their corresponding field.

Example:

```
  `add a column to display the first name
$Error:=AL_InsertFields (eList;File(»[People]);4;1;Field(»[People]First
                Name)
```

AL_RemoveFields

AL_RemoveFields *(AreaName; ColumnNum; NumFields)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
ColumnNum	integer	column at which to remove the first field
NumFields	integer	number of fields to remove (up to 255)

AL_RemoveFields is used to remove fields from AreaList Pro. *NumFields*, beginning at *ColumnNum*, will be removed from the list.

All subsequent columns will maintain their settings. In other words, any header text, column styles, etc. will stay with their corresponding field.

Example:

```
  `remove two columns, beginning at column #4
$Error:=AL_RemoveFields (eList;4;2)
```

AL_UpdateFields

AL_UpdateFields *(AreaName; UpdateMethod)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
UpdateMethod	integer	method to use to update the AreaList Pro object

AL_UpdateFields is used to update AreaList Pro. Use this command whenever any records of the fields being displayed are changed (records added, deleted, or modified), but the fields themselves remain the same.

AL_UpdateFields must be called after modifying the fields and before any other setup commands (sorting, formatting, etc.).

UpdateMethod — Integer. This parameter tells AreaList Pro how to update the AreaList Pro object AreaName. The possible values are:

Value	Description	When to Use
0	Refresh the AreaList Pro object, but don’t update any records, and don't recalculate any values.	When changes are made to formatting, color, styles, etc.

Value	Description	When to Use
1	Refresh the AreaList Pro object, and update the visible records, but don't recalculate any values.	When changes are made to the contents of the records shown in the visible rows.
2	Rescan all visible rows and recalculate all applicable heights, widths, and other related values. The scroll position, and row or cell selection will be reset.	If column or row resizing is necessary, or you have added or deleted records pertaining to the displayed fields. Also if you show or hide either scroll bar, the headers, or footers.

AL_SetSubSelect

AL_SetSubSelect *(AreaName; FirstRecord; NumRecords)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
FirstRecord	longint	the first record to display
NumRecords	longint	the number of records to display

AL_SetSubSelect is used to tell AreaList Pro to display a different subselection of records from the current selection. This command will have the same effect on the AreaList Pro object as calling ***AL_UpdateFields*** (page 101) with *UpdateMethod* set to 2, in addition to changing the subselection of records to be displayed. Thus if this command is called, there is no need to also call ***AL_UpdateFields***.

FirstRecord — Longint. This parameter is used to set the first record in the selection to be displayed in the AreaList Pro object. If *FirstRecord* is greater than or equal to the number of records in the selection, then it will be set to the last record in the selection. The default is 1.

NumRecords — Longint. This parameter is used to set the number of records in the selection to be displayed in the AreaList Pro object. The possible values are:

>= 0	Display this number of records.
-1	Display 32,750 records or the number of records from *FirstRecord* until the end of the selection, whichever is less.

If *NumRecords* is greater than 32,750 records and there are more than 32,750 records from *FirstRecord* until the end of the selection, then *NumRecords* will be set to 32,750. If *NumRecords* is greater than the number of records from *FirstRecord* until the end of the selection, then *NumRecords* will be set to the number of records from *FirstRecord* until the end of the selection.

If this command is not called, then *FirstRecord* will be set to 1 and *NumRecords* will be set to 32,750 records or the number of records in the selection, whichever is less.

Example:

```
  `set up the eList AreaList Pro object to display 10,000 records beginning at
                    record 5001
AL_SetSubSelect (eList;5001;10000)
```

Enterability Commands

Initiating Data Entry

The method for initiating entry to a cell, and for selecting rows, is set with the *EntryMode* parameter of ***AL_SetEntryOpts*** (page 119). Initiating entry can be done in any one of eight different ways, each of which also determines the method for selecting rows. See "***AL_SetEntryOpts***" on page 119 for complete information.

Entering Data

The capability to edit data during typed data entry is initiated automatically, and no programming is necessary to invoke these functions.

When data entry is initiated on an AreaList Pro cell, the array contents for the element corresponding to that cell are copied to the zero element of the same array. Since this element is usually never used, it makes a convenient storage place for the data in case you wish to revert to the old value; however, you should take care not to use this zero array element elsewhere in your code while data entry is in progress.

Note: When fields are displayed you are responsible for saving the contents of the field.

Two commands, ***AL_SetCellHigh*** (page 125) and ***AL_GetCellHigh*** (page 126), can be used to set the highlighted range of characters in the data entry cell, or get the range of characters highlighted by the user, respectively. ***AL_SetCellHigh*** can also be used to set the insertion point between two characters. After the user ends data entry on a particular array element, ***AL_GetCellMod*** (page 125) can be used to determine if the data has been altered. ***AL_GetCellMod*** and ***AL_GetCellHigh*** can only be used within an entry finished callback procedure. See "Using Callback Procedures During Data Entry" on page 108.

Other programmable data entry specifications include the use of the Return key for movement during data entry, or insertion of a Carriage Return character into the text data being entered. This is controlled using the *AllowReturn* parameter of ***AL_SetEntryOpts*** (page 119) *Please read the section "Moving* *the Current Entry Cell" on page 107 for more information.*

You can also specify that seconds be displayed (hh:mm:ss) when the user is entering time data through the use of the *DisplaySeconds* parameter of ***AL_SetEntryOpts*** (page 119).

For boolean data type arrays, two data entry methods can be specified: a checkbox or radio buttons. ***AL_SetEntryCtls*** (page 121) is used to specify which of these controls is used, and to which column it applies.

Filters

In order to use data entry filters ***AL_SetFilter*** (page 116) must be used on a per column basis. Standard 4D filter strings can be used, except that place holders are not supported and will be ignored. *Pre-defined styles may not be used for data entry filters.*

Maximum Length of a String Exceeded

As mentioned in "Entering Data" on page 105, when the maximum string length is exceeded during data entry on a string array, the system beep will sound for each character typed past the string length, and the character will be ignored. However, because of the way in which the 4th Dimension compiler allocates memory for string array elements, this may not always be true: sometimes an extra character can be entered, and it will be stored with the array element. If the database is compiled with Range Checking enabled, this results in the unpleasant side effect that a range check error will occur when this element's value is accessed in 4D code.

A 4D string is stored in memory as a Pascal string, which is a length byte followed by bytes storing each of the characters. The compiler will only allocate an even number of bytes. This means that for a string of length 7, 4D allocates 8 bytes (1 for the length and 7 for the character data.) However, if the string was declared to be 8 characters long, 4D would allocate 10 bytes of memory (1 length byte, 8 character bytes, and 1 additional byte to achieve an even number.) It is this extra byte which causes the problem, because 4D will allow information to be stored in it, but will later perform error checking and generate a runtime error.

AreaList Pro cannot determine the declared length of an array element; it can only detect how long the element actually is. Thus, there is no way to prevent the user from entering the extra character. There are two workarounds available for this problem.

The first is to avoid it by declaring all string arrays to be an odd number of characters in length. This will prevent the extra byte from being allocated, and the extra character from being entered. The second workaround is to compile the database with the Range Checking option turned off.

Popups

As an alternative to typed data entry, you can specify that a column use popup menus by using the *PopupArray* parameter of ***AL_SetEnterable*** (page 114). In this parameter, an array is passed to AreaList Pro, with which AreaList Pro will build a popup menu.

No array need be passed to AreaList Pro for a time or date column which uses a popup menu — AreaList Pro provides specialized menus for these data types. The presence of a popup menu in a cell does not prohibit the user from entering typed data; the *Enterable* parameter of ***AL_SetEnterable*** (page 114) allows you to control whether either one or both of these data entry methods are allowed.

Note: The popup menu array must be of the same data type as the data in the column. It is important that the array used for a popup not be disposed of until it is no longer needed.

Note: ***AL_SetEnterable*** *(page 114) must be called when any changes are made to a PopupArray.*

Moving the Current Entry Cell

The action of the Carriage Return key is determined by the programmer using the *AllowReturn* parameter of ***AL_SetEntryOpts*** (page 119), depending upon the data entry requirements of the database.

The user’s ability to control movement while in data entry can also be established with the use of the *MoveWithArrows* and *MapEnterKey* parameters of this command. The *MoveWithArrows* parameter will allow the user to move from cell to cell while in data entry using the four Arrow keys. *MapEnterKey* enables you to cause the Enter key to act the same way as either the Tab key or the Return key. When using this parameter, it should be noted that the Enter key is often used in 4th Dimension for other functions which may conflict with its AreaList Pro meaning.

A variety of AreaList Pro commands enable you to monitor and control movement during data entry. The current and previous data entry cells can be determined by using ***AL_GetCurrCell*** (page 124) and ***AL_GetPrevCell*** (page 124), respectively. Movement from cell to cell, while staying in data entry mode, can be accomplished using ***AL_GotoCell*** (page 126). ***AL_SkipCell*** (page 127) can be used in the entry started callback procedure to cause data entry on a particular cell to be skipped. Data entry can be terminated via ***AL_ExitCell*** (page 128).

Using Callback Procedures During Data Entry

A “callback” is a global 4D procedure which is executed by an external. AreaList Pro lets you make use of callbacks when entering and exiting a cell, and when a popup menu is clicked or released. This feature provides you with considerable control over user actions, allowing you to do such things as reject an entry, provide a choice list, or simply skip a particular cell.

Your callback procedures may use any 4D commands, but can only use the AreaList Pro commands shown in Tables 1 and 2 below.

Table 3: Enterability Commands Allowed from a Callback

AL_ExitCell	***AL_GetCellMod***	***AL_GetCurrCell***
AL_GetPrevCell	***AL_GotoCell***	***AL_SetCallbacks***
AL_SetEnterable	***AL_SetEntryOpts***	***AL_SetFilter***
AL_SetEntryCtls	***AL_SetCellHigh***	***AL_GetCellHigh***
AL_SkipCell		

Table 4: Other AreaList Pro Commands Allowed from a Callback

AL_GetColLock	***AL_GetScroll***	***AL_GetSort***
AL_GetWidths	***AL_SetBackClr***	***AL_SetForeClr***
AL_SetFormat	***AL_SetHdrStyle***	***AL_SetHeaders***
AL_SetRowColor	***AL_SetRowStyle***	***AL_SetStyle***
AL_SetFooters	***AL_SetFtrStyle***	***AL_UpdateArrays***
AL_UpdateFields		

Note: ***AL_UpdateArrays*** *(page 46) can only be called with UpdateMethod equal to -1 and* ***AL_UpdateFields*** *(page 101) can only be called with UpdateMethod equal to 0 or 1 from a callback procedure.*

In addition to altering the array content, you can change color and style, reject or accept entered data, and change the current data entry cell using the AreaList Pro commands listed above. You should not call any command which changes the number of displayed arrays, their position in the area, or their sorted order.

Executing a Callback Upon Entering a Cell

An “entry started” callback procedure is a 4th Dimension procedure called when data entry begins for a cell or an AreaList Pro popup menu is clicked, and is specified by passing the procedure name in the *EntryStartedProc* parameter of ***AL_SetCallbacks*** (page 117). If this parameter is a null string then no procedure will be called.

AreaList Pro will pass the entry started callback procedure two parameters if arrays are being displayed, or three parameters if fields are displayed. The first parameter is a long integer that corresponds to the name of the AreaList Pro object on the layout. The second parameter is a long integer that reports what action caused data entry to begin in the cell. The third parameter is a long integer that reports whether the record was loaded or not (when fields are being displayed).

You must use the declaration

C_LONGINT($1;$2;$3)

in your callback procedure. Since the first parameter, the long integer $1, contains 4D’s representation of the AreaList Pro object, it can be used as the first parameter of any AreaList Pro procedure called. As stated above, the second parameter passed to the callback routine, the long integer $2, contains the method by which data entry began, according to the following table:

Value	EntryMethod
1	Click in Cell
2	Tab
3	Shift-Tab
4	Return
5	Shift-Return
6	***AL_GotoCell***
7	not used

Value	EntryMethod
8	not used
9	***AL_SkipCell***
10	Click on cell popup when cursor not already in cell
11	Click on cell popup when cursor already in cell

The entry started callback is also executed whenever a popup menu is clicked, but before the menu is actually displayed. When this occurs, the *EntryMethod* provided by AreaList Pro will be 10 if the popup was clicked on a cell other than the one actively in data entry. Method 11 will be reported if data entry was already established in the cell for which the popup was clicked. One of the primary uses of the entry started callback when the popup is clicked would be to load the array from which the popup is built, then use ***AL_SetEnterable*** (page 114) to pass the array to AreaList Pro.

If the third parameter is 1, then the record was loaded properly and the field contents can be edited. If the third parameter is 0, then the record is locked by another process or user. If typed data entry is underway and the record can not be loaded, then ***AL_GotoCell*** (page 126) or ***AL_SkipCell*** (page 127) may be used to continue data entry in another cell. If neither of these commands is called then data entry will end. If popup data entry is underway and the record can not be loaded then data entry will end.

Executing a Callback Upon Leaving a Cell

An “entry finished” callback procedure is a 4th Dimension procedure called when data entry ends for a cell, or when an AreaList Pro popup menu is released for a cell not in typed data entry. The entry finished callback procedure is specified by passing the procedure name in the *EntryFinishedProc* parameter of ***AL_SetCallbacks*** (page 117). If this parameter is a null string then no procedure will be called.

The entry finished callback procedure is passed two parameters by AreaList Pro. The first parameter is a long integer that corresponds to the name of the AreaList Pro object on the layout. The second parameter is a long integer that reports what action caused data entry to end in the cell.

You must use the following declaration in your callback procedure.
C_LONGINT($1;$2)

Since the first parameter, the long integer $1, contains 4D’s representation of the AreaList Pro object, it can be used as the first parameter of any AreaList Pro procedure called. As stated above, the second parameter passed to the callback routine, the long integer $2, contains the method by which data entry ended, according to the following table:

Value	ExitMethod
1	Click outside cell on object
2	Tab
3	Shift-Tab
4	Return
5	Shift-Return
6	***AL_GotoCell***
7	***AL_ExitCell***
8	Deselect the cell
9	not used
10	Cell popup released when cursor not already in cell
11	Cell popup released when cursor already in cell

The entry finished callback procedure is actually a function. It must return **True** for the value entered into the cell to be accepted, and **False** for the value to be rejected. If the value is rejected the user will not be allowed to leave the cell. See the 4th Dimension Language Reference for more details about functions and procedures.

The entry finished callback function is also called when a popup menu is released. In this case, the *ExitMethod* reported by AreaList Pro to the callback will be 10 if typed data entry was in progress in the cell which contains the popup, or 11 if typed data entry was not in progress in that cell. ***AL_GotoCell*** (page 126) can be used to establish typed data entry on the cell if it did not exist before the popup was clicked.

Note: If typed data entry is already established for the cell in which the popup exists, the entry finished callback function will not run when the popup menu is released.

When displaying arrays and data entry is initiated in a cell, the contents of the array element will be copied into the zero element of the array being displayed in the column. *Please read the section "Initiating Data Entry" on page 105 for more information.*

When fields are displayed, the contents of the field are not copied. Thus it is up to you to save the field contents in the entry started callback procedure if they will be needed for comparison in the entry finished callback procedure.

When displaying arrays and the entry finished callback procedure is executed, the array element corresponding to the cell has already been updated with the new value that was entered by the user. Thus, the zero element which contains the old data and the element representing the current cell can both be used to determine data validity.

Among the possible situations and responses that may occur are the following:

- The data is valid. Set $0:=**True** to complete data entry for the cell.
- The data is invalid. Copy the old data from the zero element to the array element corresponding to the cell. Set $0:=**True** to complete data entry for the cell.

For example:

```
aFname{vRow}:=aFname{0}    `Reset the cell contents to their original state
$0:=True
```

- The data is invalid. Inform the user that the data is invalid. Set $0:=**False** to force the user to remain in the cell and enter another value.
- The data is invalid. Inform the user that the data is invalid. Modify the cell contents, call ***AL_GotoCell*** (page 126) to go to the current cell, and set $0:=**True**. This achieves the same effect as rejecting the entry, but allows the cell contents to be modified.

For example:

```
Fname{vRow}:=aFname{0}    `Reset the cell contents to their original state
AL_GotoCell(eList;vColumn;vRow)  `go to the same cell
$0:=True
```

Notifying the User of Invalid Data Entry from the Exit Callback

You can flag invalid data by returning **False** from the exit callback procedure after data entry on a cell. AreaList Pro, in turn, will tell 4D to remain in the external area. However, due to a bug in 4D v3, the second attempt to refuse the data entered in the cell is ignored by 4D, and the external area is deselected.

To workaround this 4D bug, add code to the exit callback procedure to open a window (Any window will work, including an **Alert**, **Confirm**, **Request**, etc.).

```
OPEN WINDOW(1;1;2;2;1)
CLOSE WINDOW
```

This code, when added to the callback procedure will avoid the problem, and will have an almost unnoticeable effect on the display of the layout.

Redrawing the Display from the Callback Procedure

You may want to display a variable which has been updated in one of the available callback procedures on the same layout as the AreaList Pro object. The variable's value will be successfully updated in the callback procedure, but it will not be displayed on the layout immediately. This is because 4D will not refresh the screen when a displayed value changes while an external is controlling execution.

In 4D version 3, a mechanism has been provided to allow you to force such a screen update. If the command **CALL PROCESS** is used with the process id parameter = -1, 4D will refresh all windows displaying an interprocess variable. This method requires that if a variable is updated from the callback procedure, then that variable must be an interprocess variable. In addition, the **CALL PROCESS** command should be executed from the callback procedure.

Example:

```
C_LONGINT($1;$2;$AL_Object;$Action;$i)
C_REAL($Total)
$AL_Object:=$1
$Action:=$2
$Total:=0
For($i;1;Size of array(aAmounts))
  $Total:=$Total+aAmounts{$i}
End for
◊Total:=$Total
CALL PROCESS(-1)
```

```
$0:=True
```

Unfortunately, this effect cannot be achieved in earlier versions of 4D. Any variables displayed on the layout which are updated in a callback must wait for some other event to cause a screen refresh.

Exiting Data Entry

Entry mode can be terminated procedurally by using ***AL_ExitCell*** (page 128).

Commands

AL_SetEnterable

AL_SetEnterable*(AreaName;ColumnNum;Enterable;PopupArray; MenuSetReference)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
ColumnNum	integer	column to apply enterability
Enterable	integer	enterability mode
PopupArray	array	4D array to display in popup menu
MenuSetReference	longint	Reference to a MenuSet™ menu

AL_SetEnterable is used to set the enterability of a column.

Enterable — Integer. This parameter specifies the methods of enterability for *ColumnNum*.

0	Not enterable
1	Enterable using typed characters only (default)
2	Enterable using popup menu only
3	Enterable using both typed characters and popup menu

ColumnNum — Integer. This parameter specifies what column to act on. If *ColumnNum* is 0, then all columns will be affected.

PopupArray — Array, integer, longint, real, string or text. This array will be displayed in the popup menu and must be the same type as the array or field displayed in *ColumnNum*. If it is not the same type or if it has no elements, then a menu containing a single disabled menu item with the text “No items in this menu” will be displayed.

An array is not needed to display a time or date popup menu; built-in menus are provided. Columns containing boolean or picture arrays can not contain popup menus.

Note: Do not dispose of the array in 4D until the popup is no longer needed.

MenuSetReference — Longint. This parameter passes the reference obtained from the MenuSet™ command ***MS_ShareIPMenu***. This parameter is used to install a MenuSet menu in the column represented by *ColumnNum*.

A MenuSet menu may be installed in a column containing an array or field of one of the following types: integer, longint, real, string or text.

If this parameter is passed, then the MenuSet menu will be used. If this parameter is not passed, then the values in *PopupArray* will be displayed in an AreaList Pro popup.

Note: When the user selects an item from the MenuSet menu, the entry finished callback procedure is run. In this callback the appropriate MenuSet commands must then be called to determine the user's selection. Then the user's selection must be placed in the array element corresponding to the cell entered.

If this command is called in the **During** phase, use ***AL_UpdateArrays*** (page 46) or ***AL_UpdateFields*** (page 101) to redraw the AreaList Pro object as needed. See “Using Callback Procedures During Data Entry” on page 108 for a discussion of using callback procedures with popup menus.

If this command is not called, then all columns will be enterable using typed characters only.

Examples:

```
AL_SetEnterable(eArea;4;1)  `set column 4 to be enterable using typed char-
                    acters only
AL_SetEnterable(eArea;0;0)  `set all columns to be not enterable
AL_SetEnterable(eArea;3;2;aProducts)  `set the third column to be enterable
                    via a popup menu containing the items in the array aProducts

ARRAY STRING(40;aTeam1;10)
ARRAY STRING(40;aTeam2;10)
ARRAY STRING(10;aTimes;10)
ARRAY STRING(10;aFields;10)
C_LONGINT(ipTeams;msMenu)

`make MenuSet I/P menu
```

```
ipTeams:=MS_MakeIPMenu ("15000";"FWASA Teams")
$SubID:=MS_SetSubMenu (ipTeams;0;1;15001)
$SubID:=MS_SetSubMenu (ipTeams;0;2;15002)
$SubID:=MS_SetSubMenu (ipTeams;0;3;15003)
$SubID:=MS_SetSubMenu (ipTeams;0;4;15004)

msMenu:=MS_ShareIPMenu (ipTeams)   ` get the MenuSet reference

ARRAY STRING(2;aDummy;0)
AL_SetEnterable (eArea;3;2;aDummy;msMenu)   ` use the MenuSet menu in
                  column 3
```

AL_SetFilter

AL_SetFilter*(AreaName;ColumnNum;EntryFilter)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
ColumnNum	integer	column to apply entry filter
EntryFilter	string	filter for input data

AL_SetFilter is used to set the entry filter for a column.

ColumnNum — Integer. This parameter specifies the column to act on. If *ColumnNum* is 0, then all columns will be affected.

EntryFilter — String. This parameter specifies the filter to use. Entry filters will function as they do in 4th Dimension, except that they will not handle placeholders. Predefined styles may *not* be used.

Please read the section “Filters” on page 106 for more information.

Examples:

```
AL_SetFilter(eList;3;"&9")  `column 3, allow numbers
AL_SetFilter(eList;6;"~a")  `column 6, allow lower and uppercase, make all
                  uppercase

CapsFilter:="~"+Char(34)+"A-Z;a-z;0-9; ;.;0;/;*;(;);&;$;\;"+Char(34)
AL_SetFilter(eList;4;CapsFilter)   `column 4, allow multiple groups and sev-
                  eral individual characters
```

AL_SetCallbacks

AL_SetCallbacks*(AreaName;EntryStartedProc;EntryFinishedProc)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
EntryStartedProc	string	4D procedure called when entry in cell started
EntryFinishedProc	string	4D procedure called when entry in cell finished

AL_SetCallbacks is used to set callback procedures that are used with data entry. *Please read the section "Using Callback Procedures During Data Entry" on page 108 for more information.*

EntryStartedProc — String. This procedure will be called whenever data entry is started in a cell or when a popup menu is clicked. If this is an empty string then no procedure will be called.

The *EntryStartedProc* is passed two parameters. The first parameter is a longint that corresponds to the name of the AreaList Pro object on the layout. The second parameter is a longint that reports what action caused data entry to be started in the cell. For a list of the possible values of the second parameter, see the table below.

Value	Entry Method
1	Click in Cell
2	Tab
3	Shift-Tab
4	Return
5	Shift-Return
6	***AL_GotoCell***
7	not used
8	not used
9	***AL_SkipCell***
10	Other cell popup clicked
11	Active cell popup clicked

EntryFinishedProc — This procedure will be called whenever data entry is finished in a cell or when a popup menu is released in a cell for which typed data entry has not been established. This procedure must be a function. It must return **True** for the value entered into the cell to be accepted and **False** for the value to be rejected. If this is the null string then no procedure will be called.

The *EntryFinishedProc* is passed two parameters. The first parameter is a longint that corresponds to the name of the AreaList Pro object on the layout. The second parameter is a longint that reports what action caused data entry to be finished in the cell. For a list of the possible values of the second parameter, see the table below.

Value	Exit Method
1	Click outside cell on object
2	Tab
3	Shift-Tab
4	Return
5	Shift-Return
6	***AL_GotoCell***
7	***AL_ExitCell***
8	Deselect the cell
9	not used
10	Other cell popup released
11	Active cell popup released

When a cell is entered the data will be copied into the zero element of the array being displayed in the column. When the *EntryFinishedProc* is executed the array element corresponding to the cell will already be updated with the new value that was entered.

Among the possible situations and responses that may occur are the following:

- The data is valid. Set $0:=**True** to complete data entry for the cell.
- The data is invalid. Copy the old data from the zero element to the array element corresponding to the cell. Set $0:=**True**

to complete data entry for the cell.

For example:

```
aFname{vRow}:=aFname{0}    `Reset the cell contents to their original state
$0:=True
```

- The data is invalid. Inform the user that the data is invalid. Set $0:=**False** to force the user to remain in the cell and enter another value.
- The data is invalid. Inform the user that the data is invalid. Modify the cell contents, call ***AL_GotoCell*** to go to the current cell, and set $0:=**True**. This achieves the same effect as rejecting the entry, but allows the cell contents to be modified.

Examples:

```
aFname{vRow}:=aFname{0}    `Reset the cell contents to their original state
AL_GotoCell(eList;vColumn;vRow)  `go to the same cell
$0:=True

  `don't install an entry started procedure, do install an entry finished procedure
AL_SetCallbacks(eNames;"";"EntryDoneProc")
```

AL_SetEntryOpts

AL_SetEntryOpts*(AreaName;EntryMode;AllowReturn;DisplaySeconds; MoveWithArrows; MapEnterKey)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
EntryMode	integer	method to initiate entry
AllowReturn	integer	allow entry of carriage returns into text arrays
DisplaySeconds	integer	display seconds in time arrays during data entry
MoveWithArrows	integer	move enterable cell using the Arrow keys
MapEnterKey	integer	map the enter key to function as another key

AL_SetEntryOpts is used to control several AreaList Pro options pertaining to data entry. *Please read the section “Moving the Current Entry Cell” on page 107 for more information.*

EntryMode — Integer, 0 to 7. This option determines the method that the user can use to initiate data entry and select rows with the mouse. The table below describes the possible values. The default is 1.

Value	Entry	Selection
0	None	Single-click
1	None	Single and Double-click
2	Single-click	None
3	Double-click	Single-click
4	<Cmd> Double-click	Single and Double-click
5	<Shift> Double-click	Single and Double-click
6	<Option> Double-click	Single and Double-click
7	<Control> Double-click	Single and Double-click

AllowReturn — Integer, 1 or 0.

1 the user can enter a carriage return character into a text array element,
0 the carriage return character will move the enterable cell as described in the "Moving the Current Entry Cell" on page 107 (default)

DisplaySeconds — Integer, 1 or 0.

1 seconds will be displayed in time array elements during data entry
0 seconds will not be displayed (default)

MoveWithArrows — Integer, 1 or 0.

1 the Arrow keys will move the enterable cell to the next cell according to the key pressed
0 the Arrow keys will move the insertion point within the enterable cell (default)

MapEnterKey — Integer.

0 Do not map the Enter key (default)
1 Map the Enter key to act like the Tab key
2 Map the Enter key to act like the Return key

Note: The Enter key is in many cases used to accept a record or *perform some other action in 4D. If the Enter key is not acting as expected, make sure that it is not being used as a key equivalent somewhere on the layout.*

Examples:

```
  `initiate data entry with a double-click, single-click selection,
  `don't allow carriage return characters to be entered into text arrays,
  `don't display seconds in time arrays during data entry
  `map the Enter key to act like the Tab key
AL_SetEntryOpts(eNames;3;0;0;0;1)
  `initiate data entry with a single-click, no selection,
  `allow carriage return characters to be entered into text arrays,
  `display seconds in time arrays during data entry
  `use Arrows to navigate between cells
AL_SetEntryOpts(eNames;2;1;1;1;0)
```

AL_SetEntryCtls

AL_SetEntryCtls*(AreaName;ColumnNum;ControlType)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
ColumnNum	integer	column in which control appears
ControlType	integer	type of control

AL_SetEntryCtls is used to specify which type of control will be used for data entry in a column displaying a boolean array. If the column contains any other type of array, this command will be ignored.

ColumnNum — Integer. This parameter specifies the column to act on.

ControlType — Integer.

0	checkbox without title (default)
1	checkbox with title (the title is the **True** label specified in ***AL_SetFormat***)
2	radio buttons (**True** and **False** labels are specified in ***AL_SetFormat***)

If ***AL_SetEntryCtls*** is not called, then a checkbox without a title will be used for boolean data entry.

Examples:

```
  `use a checkbox with with title for data entry in column 2
AL_SetEntryCtls(eNames;2;1)
```

AL_SetCellEnter

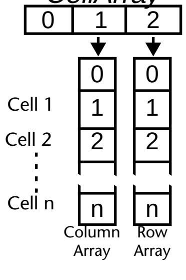
AL_SetCellEnter*(AreaName;Cell1Col;Cell1Row;Cell2Col;Cell2Row;CellArray;Enterabl e)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
Cell1Col	integer	first cell column
Cell1Row	integer	first cell row
Cell2Col	integer	last cell column
Cell2Row	integer	last cell row
CellArray	array	discontiguous cells
Enterable	integer	enterability

AL_SetCellEnter is used to set the enterability of a specific cell, range of cells, or list of cells.

- **To specify a single cell.** If *Cell1Col* and *Cell1Row* are greater than 0 and *Cell2Col* or *Cell2Row* are less than or equal to 0 then only [*Cell1Col*, *Cell1Row*] will be set.
- **To specify a range of cells.** If *Cell1Col* and *Cell1Row* are greater than 0 and *Cell2Col* and *Cell2Row* are greater than 0 then the range of cells from [*Cell1Col*, *Cell1Row*] to [*Cell2Col*, *Cell2Row*] will be set.
- **To specify discontiguous cells.** If *Cell1Col* or *Cell1Row* are less than or equal to 0 then the cells in *CellArray* will be set.

CellArray — two-dimensional integer array. The first dimension *must* be two. The first array is for the column indices and the second array is for the row indices. The second dimension must be the same as the number of cells that are to be selected. See the following illustration.



Enterable— Integer, 1, 0 or -1.

1 the cell is enterable by typing
0 the cell is not enterable by typing

-1	remove any cell-specific enterability which has been set for the cells

The *MoveWithData* option of ***AL_SetCellOpts*** (page 65) controls whether cell enterability stays with a cell whenever sorting, row dragging, or column dragging occurs.

Examples:

```
ARRAY INTEGER(aInt;2;0)

  `set the cell in the third column, first row, to be enterable
AL_SetCellEnter(eList;3;1;0;0;aInt;1)

  `set the cells in the fourth row (ten columns) to be non-enterable
  `the row number is the same for all the cells, just the column number
  `changes. So specify a range of values.
AL_SetCellEnter(eList;1;4;10;4;aInt;0)

  `set the cells in rows 8, 9, and 10, the first two columns, to be non-enterable
AL_SetCellEnter(eList;1;8;2;10;aInt;0)
```

AL_GetCellEnter

AL_GetCellEnter*(AreaName;CellCol;CellRow;Enterable)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
CellCol	integer	cell column
CellRow	integer	cell row
Enterable	integer	enterability

AL_GetCellEnter is used to determine if the enterability of the specified cell has been explicitly set with ***AL_SetCellEnter***. Note that ***AL_GetCellEnter*** will not get the column enterability.

Enterable— Integer, 1, 0 or -1.

1	the cell is enterable by typing
0	the cell is not enterable by typing
-1	the cell’s enterability has not been previously set

AL_GetCurrCell

AL_GetCurrCell*(AreaName;ColumnNum;RowNum)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
ColumnNum	integer	column where entry cell is located
RowNum	integer	row where entry cell is located

AL_GetCurrCell will return the currently enterable cell. This command is only valid from a callback procedure. *Please read the section "Using Callback Procedures During Data Entry" on page 108 for more information.*

ColumnNum — Integer. This parameter returns the current cell's column number.

RowNum — Integer. This parameter returns the current cell's row number.

AL_GetCurrCell will return 0 in both the *ColumnNum* and the *RowNum* if there is not a cell being entered. Both *ColumnNum* and *RowNum* must be global variables.

Example:

AL_GetCurrCell(eArea;vColumn;vRow) `get the current cell

AL_GetPrevCell

AL_GetPrevCell*(AreaName;ColumnNum;RowNum)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
ColumnNum	integer	column where entry cell was located
RowNum	integer	row where entry cell was located

AL_GetPrevCell will return the previously enterable cell.

ColumnNum — Integer. This parameter returns the previous cell's column number.

RowNum — Integer. This parameter returns the previous cell's row number.

AL_GetPrevCell will return 0 in both the *ColumnNum* and the

RowNum if there was not a cell being entered previously. Both *ColumnNum* and *RowNum* must be global variables.

Example:

```
AL_GetPrevCell(eArea;vColumn;vRow)  `get the previous cell
```

AL_GetCellMod

AL_GetCellMod*(AreaName) ➞ CellModified*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
CellModified	integer	was the cell modified?

AL_GetCellMod will report whether or not the contents of the cell have been modified. Use this command in the *EntryFinishedProc* callback. *Please read the section "Executing a Callback Upon Leaving a Cell" on page 110 for more information.*

CellModified — Integer. This parameter reports whether or not a cell was modified.

0	not modified
1	modified

Example:

```
If(AL_GetCellMod(eList)=1)  `was the value modified?
  AL_GetCurrCell(eList;vCol;vRow)
  If(vCol=5)  `5th column is Line Item quantity
    aExtended{vRow}:=aQty{vRow}*aPrice{vRow}
    AL_UpdateArrays(eList;-1)
  End if
End if
```

AL_SetCellHigh

AL_SetCellHigh*(AreaName;StartPosition;EndPosition)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
StartPosition	integer	first character of cell text to highlight
EndPosition	integer	last character of cell text to highlight

AL_SetCellHigh will highlight a range of characters within a cell, from *StartPosition* to *EndPosition*-1. When *StartPosition* = *EndPosition*, then the insertion point will be positioned prior to the character indicated in *StartPosition*, and none of the characters in the cell will be highlighted.

Example:

```
  `entry finished callback:
 If(Not(vDataValid))
   AL_SetCellHigh(eArea;vStart;vEnd)  `highlight the cell contents to indicate
                     error
 End if
```

AL_GetCellHigh

***AL_GetCellHigh**(AreaName;StartPosition;EndPosition)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
StartPosition	integer	first character of highlighted cell text
EndPosition	integer	last character of highlighted cell text

AL_GetCellHigh will obtain the highlighted range of characters within a cell. This command may be used to provide user feedback after performing error checking on entered data, and can only be used in the entry finished callback procedure. *Please read the section "Executing a Callback Upon Leaving a Cell" on page 110 for more information.*

See "***AL_SetCellHigh***" on page 125.

StartPosition — Integer. This parameter indicates the first highlighted character.

EndPosition — Integer. This parameter indicates the last highlighted character.

AL_GotoCell

***AL_GotoCell**(AreaName;ColumnNum;RowNum)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
ColumnNum	integer	column to move entry to
RowNum	integer	row to move entry to

AL_GotoCell will place the cursor into the specified cell. If the cell does not exist or has been set to not enterable by ***AL_SetEnterable*** (page 114), then this command will have no effect.

*If you use **AL_GotoCell** from a script or procedure other than the entry or exit callback procedure, you must precede it with the 4D command* **GOTO AREA**. *This is because **AL_GotoCell** only works if the AreaList Pro object is selected.*

If ***AL_GotoCell*** is called in the **Before** phase, the AreaList Pro area must be the first in the entry order for the layout.

ColumnNum — Integer. This parameter specifies the cell's column.

RowNum — Integer. This parameter specifies the row's column.

Example:

```
 `Entry Callback Procedure
AL_GetCurrCell(eItems;vCol;vRow)
If(vCol=3)  `unit price
  If(gAccess#"Sales")  `does user have security access to this field?
    If($2=2)  `tab
      AL_GotoCell(eItems;vCol+1;vRow)  `goto the next cell
    Else   `not Tab
      AL_ExitCell(eItems)  `end data entry
    End if
  End if
End if
```

AL_SkipCell

AL_SkipCell*(AreaName)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout

AL_SkipCell will skip the current data entry cell and proceed to the next cell. This command can only be called from the entry started callback procedure. *Please read the section "Executing a Callback Upon Entering a Cell" on page 109 for more information.*.

If data entry in a cell is begun via a Tab, Shift-Tab, Return, Shift-Return, or click, then ***AL_SkipCell*** moves data entry to the next appropriate cell, according to the entry method.

If the cell was entered via a mouse click, the cell will be exited and data entry will be ended.

If the cell was entered because of ***AL_SkipCell*** called from the previous cell, then data entry will similarly be moved to the next cell.

If the method by which data entry begun is anything else, this command will be ignored.

Example:

```
  `Entry Callback Procedure
AL_GetCurrCell(eItems;vCol;vRow)
If(vCol=3)  `unit price
  If(gAccess#"Sales")  `does user have security access to this field?
    AL_SkipCell(eItems)  `goto the next cell or end data entry
  End if
End if
```

AL_ExitCell

AL_ExitCell*(AreaName)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout

AL_ExitCell will exit the currently enterable cell. If there is not a cell being entered then ***AL_ExitCell*** will have no effect.

AL_ExitCell does not need to be used to deselect a cell undergoing data entry if

- a menu is selected
- another layout object is clicked
- the User clicks elsewhere on the AreaList Pro object.

These cases will all terminate data entry normally without the use of this command, and the cell will receive its normal exit callback. ***AL_ExitCell*** is required, however, to terminate data entry from an entry callback procedure.

Example:

```
  `Entry Callback Procedure
  `don't allow entry into cell at column 3, row 4
AL_GetCurrCell(eList;vCol;vRow)
If((vCol=3) & (vRow=4))
  AL_ExitCell(eList)
End if
```

Dragging Commands

Background

AreaList Pro v5.1 utilizes the Macintosh Drag Manager, which is included in System 7.5 and can be installed into earlier versions of System 7.x. It can not be installed into System 6.0.x.

To install the Macintosh Drag and Drop Extension

1. Drag the extension onto the System Folder.
   A confirm dialog will appear asking you if you wish to install it into the Extensions folder.
2. Click OK.
3. Restart the computer.

When AreaList Pro is used on a Macintosh which does not have the Macintosh Drag and Drop extension installed, the new dragging capabilities (version 5.1 and later) will not be available.

To enable dragging in the Windows version of AreaList Pro, you must have the W4D_Drag.DLL file installed. See “Installing the Windows Version of AreaList Pro” on page 7 for more information.

You can use ***AL_DragMgrAvail*** (page 134) to determine if the Macintosh Drag and Drop extension is installed.

Up to 100 AreaList Pro and or DropArea objects may be active and be dragged from or accept drags from other objects.

AreaList Pro versions prior to 5.1

The older method of dragging in AreaList Pro (pre-version 5.1) is still supported. However, it is not available for a particular object if the new commands are used.

An AreaList Pro object can use the new method of dragging (the Macintosh Drag Manager) or the old method, but not both. If you’ve used one of the commands in this chapter, any of the old commands are disabled (ignored), until you “turn off” the new commands. The new commands, documented in this chapter, are the preferred method for implementing dragging with AreaList Pro.

Please read the section “Obsolete Dragging Commands” on *page 177 for more information.*

Technical Details of the Dragging Implementation

AreaList Pro works similarly to the Drag Manager (as defined in *Inside Macintosh*) by separating the sender and receiver of a drag. The sender and receiver can be the same area, different AreaList Pro areas, or different external areas altogether. In the latter case, the external area must comply with the *4D External Drag Interface Specification* as documented by Foresight Technology, Inc.

Currently, dragging is only allowed between external areas. The user cannot drag to or from the Finder or other applications. Future versions of AreaList Pro will support the drag-and-drop being implemented in 4D version 6, provided that ACI provides the necessary external API.

You must configure AreaList Pro to allow dragging out of and into an AreaList Pro area. Commands provide the control necessary to allow dragging within an area, between two or more areas, and to not allow dragging between certain areas.

To allow dragging *out of* AreaList Pro, you must pass an access "code" for the type of data that is to be dragged. You must specify the type of data to allow to be dragged and at least one code to enable dragging, using ***AL_SetDrgSrc*** (page 135). Up to ten codes can be passed. Allowing many codes provides for more flexibility in enabling and disabling dragging between various areas. This will be explained in more depth later.

In order to allow dragging *into* AreaList Pro, you must pass an access "code" for the type of data that can be the destination of a drag. You must specify the type of data that can receive a drag, and at least one code to enable dragging, using ***AL_SetDrgDst*** (page 136). AreaList Pro supports dragging to rows, columns, and cells. As with ***AL_SetDrgSrc***, up to ten codes can be passed for flexibility reasons.

To enable cell dragging, the *CellSelection* option of ***AL_SetCellOpts*** (page 65) must be set to 1 or 2 (single cell selection or multiple cell selection is enabled).

To drag a cell out of an AreaList Pro object, set the *SourceDataType* parameter of ***AL_SetDrgSrc*** (page 135) to 3 (Cell).

To drag data into an AreaList Pro object and drop it as a cell, set the *DestDataType* parameter of ***AL_SetDrgDst*** (page 136) to 3

(Cell).

What are access “codes”?

The access codes that are passed in the ***AL_SetDrgSrc*** (page 135) and ***AL_SetDrgDst*** (page 136) commands are used to enable dragging between specific drag partners. These drag partners can be the same AreaList Pro area, different AreaList Pro areas, or different external areas.

When a drag takes place, the drag sender’s external code communicates its access codes to the drag receiver’s external code. The drag receiver will compare the access codes of the sender to its own codes. If any of the codes match, the drag is allowed. This mechanism allows a number of combinations between several drag partners. The following is an example of enabling the dragging of a row within the same AreaList Pro area.

Example:

```
  `enable drag events to rows within the this area
vSelfStr:=String(eList)    ` creates a unique code that only allows dragging
                    within this area
AL_SetDrgSrc(eList;1;vSelfStr)    ` row data type for source
AL_SetDrgDst(eList;1;vSelfStr)    ` row data type for destination
```

This example also shows how you can create a unique identifier that only enables dragging within the same AreaList Pro area.

NOTE: AreaList Pro will update the arrays and refresh the area if the drag is within the same area (row-to-row or column-to-column).

After a drag

When row, column or cell is dragged out of AreaList Pro, the following information is available to you:

- Notification that a drag occurred
- Which row, column or cell was dragged (index in array)
- Where the row, column or cell was dragged to (this area or another area)

When the drag is completed, the AreaList Pro script is executed. If a drag occurred, *ALProEvt* will be set to -5 if a row was dragged, -7 if a column was dragged, or -8 if a cell was dragged (see “Determining the User’s Action on an AreaList Pro Object” on page 145). Then ***AL_GetDrgSrcRow*** (page 138) or ***AL_GetDrgSrcCol*** (page 139) may be used to get the row, column or cell that was dragged.

To determine which external area was the destination of the drag, call ***AL_GetDrgArea*** (page 139). This command returns the *AreaName* (a long integer) and the process id of the destination area, which may be the same AreaList Pro area, another AreaList Pro area, or another external area. When dragging to another object, that object can either reside in the same window or on another window, which may require use of 4D's **CALL PROCESS** command to take action on the drag — *when dragging to other objects, AreaList Pro is only providing a user interface to the drag, and notifying you, the developer, that the drag has occurred. You are responsible for manipulating any arrays or other data structures.*

When an AreaList Pro area is the destination of a drag, the following information is available to you:

- The type of data that was the recipient of the drag (row, column or cell).
- The row, column or cell that was dragged to.

You must use ***AL_GetDrgDstTyp*** (page 140) to determine if the destination of the drag was a row, column or cell. If the destination was a row, ***AL_GetDrgDstRow*** (page 142) may be used to determine the destination row. If the destination was a column, ***AL_GetDrgDstCol*** (page 143) may be used to determine the destination column. If the destination was a cell, then both ***AL_GetDrgDstRow*** and ***AL_GetDrgDstCol*** are used to determine the destination cell. If the destination of the drag is an area on another window, then you must use 4D's **CALL PROCESS** command to communicate to the other process.

Note: AreaList Pro will update the arrays and refresh the area if the drag is within the same area (row-to-row or column-to-column).

*Note: Row dragging is disabled when an AreaList Pro object is in cell selection mode. See "**AL_SetCellOpts**" on page 65.*

Note: When dragging cells, there will be no automatic updating of arrays, even if the source and the destination lists are the same.

AreaList Pro on Multi-Page Layouts

You can place an AreaList Pro area on layouts that contain multiple pages. If you've configured the area to accept a drag from another area, you must enable and disable the AreaList Pro area using ***AL_SetDrgDst*** (page 136), depending on whether the area is on the current page. If the page containing the AreaList Pro area is not the current page, call ***AL_SetDrgDst*** with empty strings for the *DstCode* parameters. When the page becomes current, call ***AL_SetDrgDst*** with the actual *DstCode* values you wish to allow.

Please read the section "Changing Layout Pages" on page 38 for more information.

Note: You should always disable an AreaList Pro area which is not on the current layout page.

Multiple Row Dragging

To enable multiple row dragging, the following options must all be set as follows:

- The *CellSelection* option of ***AL_SetCellOpts*** (page 65) must be set to 0 (row selection is enabled).
- The *MultiLines* option of ***AL_SetRowOpts*** (page 59) must be set to 1 (multiple row selection is enabled).
- The *MultiRowDrag* option of ***AL_SetDrgOpts*** (page 137) must be set to 1 to enable multiple row dragging.

To get the rows that were dragged, use ***AL_GetSelect*** (page 150).

Note: When dragging multiple rows, there will be no automatic updating of arrays, even if the source and the destination lists are the same.

Drag DataType

This DataType represents the type of the drag for both the source and the destination. It is used in the commands ***AL_SetDrgSrc*** (page 135), ***AL_SetDrgDst***, (page 136)and ***AL_GetDrgDstTyp*** (page 140). These are the possible values:

1	Row
2	Column
3	Cell

DropArea

AreaList Pro includes a DropArea object, which can be used as a destination for dragged rows and columns. See “DropArea” on page 155.

AL_DragMgrAvail

AL_DragMgrAvail*(IsDragMgrPresent)*

Parameter	Type	Description
IsDragMgrPresent	integer	indicates whether Drag Manager is installed

AL_DragMgrAvail is used to determine if the Drag Manager is installed, and alert the user or take some other action.

IsDragMgrPresent — Integer, 0 or 1.

1	the Drag Manager is present on this machine, and AreaList Pro drag and drop functionality is available
0	the Drag Manager is *not* present on this machine, and AreaList Pro’s version 5.1 drag features cannot be used

This command can be called from any script or procedure, including *StartUp* and *Debut*, even if an AreaList Pro object is not displayed.

AreaList Pro utilizes the Macintosh Drag Manager, which is included in System 7.5 and can be installed into earlier versions of System 7.x. It can not be installed into System 6.0.x. When AreaList Pro is used on a Macintosh which does not have the Drag-and-Drop software installed, the dragging capabilities will not be available. *The pre-version 5.1 dragging capabilities will still be available. Please read the section “Obsolete Dragging Commands” on page 177 for more information.*

Example:

```
C_LONGINT(vDragMgr)
AL_DragMgrAvail(vDragMgr)
If(vDragMgr=0)
  BEEP
  ALERT(“The Macintosh Drag Manager is not installed, you will be unable to
                  drag rows or columns!”)
End if
```

AL_SetDrgSrc

AL_SetDrgSrc*(AreaName; SourceDataType; SrcCode1; SrcCode2; ... ; SrcCode10)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
SourceDataType	integer	type of item dragged
SrcCodeN	string	used to match drag partners

AL_SetDrgSrc is used to enable dragging out of the AreaList Pro object *AreaName*, by setting the access codes for the source of the drag. This command *must* be called before a drag is initiated (usually in the **Before** phase). *Please read the section "What are access "codes"?" on page 131 for more information.*

SourceDataType — Integer. Possible values are:

1	Row
2	Column
3	Cell

SrcCode — String (15 characters). The *SrcCode* can have any value, such as "RowDrag", "ColDrag", "DragToALP", etc.; however, it is meant to match a code passed into a potential drag partner. The drag partner will be the destination/receiver of the drag. That destination can be the same AreaList Pro area, a different AreaList Pro area, or another external object.

This code can be any value other than an empty string. Avoid using the strings "TEXT" or "PICT".

AreaList Pro performs the following logic during the actual drag. When the drag takes place, the source codes that were given in *SrcCode1*, *SrcCode2*, etc. will be communicated to the receiver of the drag. If any of the codes match, the drag is enabled.

See "What are access "codes"?" on page 131.

Example:

```
  `enable dragging a row within this area
vSelfStr:=String(eList)    ` creates a unique code that only allows dragging
                            within this area
AL_SetDrgSrc (eList;1;vSelfStr)    ` row data type for source
AL_SetDrgDst (eList;1;vSelfStr)    ` row data type for destination
```

AL_SetDrgDst

AL_SetDrgDst*(AreaName; DestDataType; DstCode1; DstCode2; ... ; DstCode10)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
DestDataType	integer	data type to be received
DstCodeN	string	access code(s) to be received

AL_SetDrgDst is used to enable dragging into the destination area, by setting the access codes. *Please read the section "What are access "codes"?" on page 131 for more information.*

This command must be called *before* a drag has occurred.

The *AreaName* parameter must be the destination (receiver) area of a drag.

DestDataType - Integer, 1 or 2.

1	Row
2	Column
3	Cell

For the data type specified by *DestDataType* (either row, column, or cell), you must specify at least 1 *DstCode* to enable receiving of that type.

DstCode - String (15 characters). The *DstCode* can be any value (other than an empty string), such as "RowDrag", "ColDrag", "ALPDrag", "PartNum", etc. Avoid using the strings "TEXT" or "PICT". Pass an empty string to disable dragging.

The code should be the same as what is passed into a potential drag partner. The drag partner will be the source/sender of the drag. The source area can be the same AreaList Pro area, a different AreaList Pro area, or another external object.

AreaList Pro performs the following logic during the actual drag. When the drag takes place, the destination codes that were given in *DstCode1*, *DstCode2*, etc. are compared to the source codes communicated by the sender of the drag. If any of the codes match, the drag is enabled.

See "Technical Details of the Dragging Implementation" on page 130.

Note: When AreaList Pro is placed on a page in a multi-page layout, be sure to disable dragging using this command when that page is not the currently shown page. Please read the section “AreaList Pro on Multi-Page Layouts” on page 132 for more information.

Example:

```
  `enable dragging a row within this area
vSelfStr:=String(eList)    ` creates a unique code that only allows dragging
                              within this area
AL_SetDrgDst(eList;1;vSelfStr)    ` row type for destination
```

AL_SetDrgOpts

AL_SetDrgOpts*(AreaName;DragRowWithOptKey;ScrollAreaSize;MultiRowDrag)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
DragRowWithOptKey	integer	drag row using the option key
ScrollAreaSize	integer	size of area that will activate scrolling
MultiRowDrag	integer	enable multiple row dragging

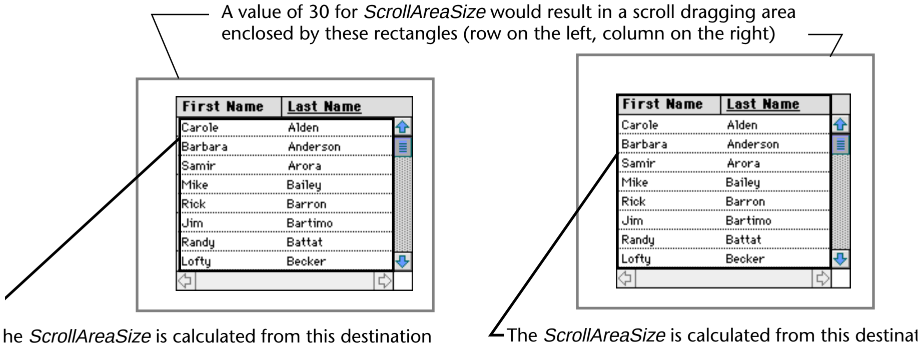
AL_SetDrgOpts is used to set various options to be used with dragging. Call this command before a drag.

DragRowWithOptKey — Integer, 1 or 0.

1	the user can drag a row by clicking on it while holding down the option key
0	the user can drag a row by clicking on it without holding down the option key (default)

ScrollAreaSize — Integer 0 to 30. This is the number of pixels outside of the destination area rectangle that will cause scrolling when the cursor is over it (see the illustrations below). If *ScrollAreaSize* is 0, then no scrolling will occur. The default is 30.

A value of 30 for *ScrollAreaSize* would result in a scroll dragging area enclosed by these rectangles (row on the left, column on the right)



First Name	Last Name
Carole	Alden
Barbara	Anderson
Samir	Arora
Mike	Bailey
Rick	Barron
Jim	Bartimo
Randy	Battat
Lofty	Becker

he *ScrollAreaSize* is calculated from this destination rea rectangle when receiving a drag into a row

The *ScrollAreaSize* is calculated from this destinat area rectangle when receiving a drag into a colum

MultiRowDrag — Integer, 0 or 1.

1	enable multiple row dragging
0	disable multiple row dragging (default)

With multiple row dragging, the arrays or records will not be automatically updated even if the source and destination lists are the same. See “Multiple Row Dragging” on page 133 for more information.

Example:

```
  `drag row without the option key, scroll in 10 pixel area, drag multiple rows
AL_SetDrgOpts(eArea;0;10; 1)
```

AL_GetDrgSrcRow

AL_GetDrgSrcRow*(AreaName;SourceRow)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
SourceRow	integer	row that was dragged

Use ***AL_GetDrgSrcRow*** to determine which row or cell was dragged after a drag has completed. The *AreaName* parameter should be the source (sender) area of a drag. This command is called from the source area’s script when *ALProEvt* = -5 (user dragged row) or *ALProEvt*=-8 (user dragged cell).

SourceRow — Integer. This parameter returns the row that was dragged.

Example:

```
  `eSrcALP script
If(During)
  C_LONGINT(vRow)
  Case of
    :(ALProEvt=-5)  ` User dragged a row
      AL_GetDrgSrcRow (eSrcALP ;vRow)
        `now do something useful
  End case
End if
```

AL_GetDrgSrcCol

AL_GetDrgSrcCol*(AreaName;SourceCol)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
SourceCol	integer	column that was dragged

Use ***AL_GetDrgSrcCol*** to determine which column or cell was dragged after a drag has completed. The *AreaName* parameter should be the source (sender) area of a drag. This command is called from the source area's script when *ALProEvt* = -7 (user dragged column) or *ALProEvt*=-8 (user dragged cell).

SourceCol — Integer. This parameter returns the column that was dragged.

Example:

```
  `eSrcALP script
If(During)
  C_LONGINT(vCol)
  Case of
    :(ALProEvt=-7)  ` User dragged a column
      AL_GetDrgSrcCol (eSrcALP ;vCol)
        `now do something useful
End case
End if
```

AL_GetDrgArea

AL_GetDrgArea*(AreaName; DestArea; DestProcessID)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
DestArea	longint	ID of the area the item was dragged to
DestProcessID	integer	process ID of the *DestArea*

Use ***AL_GetDrgArea*** to determine the destination area of the last drag. The *AreaName* parameter should be the source (sender) area of a drag. This command is called from the source area's script when *ALProEvt* = -5, -7, or -8 — user dragged a row, column or cell. *Please read the section "Determining the User's Action on an AreaList Pro Object" on page 145 for more information.*

DestArea — Long Integer. This parameter is the area reference of the area that is the destination of the drag.

DestProcessID — Integer. This parameter contains the Process ID in which the window and destination area reside. Use the 4D commands **CALL PROCESS** and **Outside call** for interprocess communication.

Note: If the DestProcessID is different from the current process, you will need to use the 4D **CALL PROCESS** *and* **Outside call** *commands to communicate to the window that contains the destination area.*

Example:

```
  `eSrcALP script
C_LONGINT(vDstArea;vDestID;vRow)

Case of
  :(ALProEvt=-5)  ` User dragged a row
    AL_GetDrgSrcRow(eSrcALP ;vRow)
    AL_GetDrgArea(eSrcALP;vDstArea;vDstID)
    If (vDstID#Current process) ` if dragged to a different process
      ◊vDstArea:=vDstArea ` store in interprocess variable
      CALL PROCESS(vDstID)
    End if
End case
```

AL_GetDrgDstTyp

AL_GetDrgDstTyp*(AreaName;DestDataType)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
DestDataType	integer	type of data which was destination of drag

AL_GetDrgDstTyp is used to determine the type of data that was the destination of the last drag. Specifically, the user may drag items to either a row, column, or a cell. After the drag has completed, ***AL_GetDrgDstTyp*** indicates whether the destination of the drag was a row, column, or a cell. The *AreaName* parameter should be the destination (receiver) area of a drag.

If the destination and source areas are actually the same area or different areas within the same process (i.e., they reside on the same layout), this command may be called from the source area's script. If the destination and source areas are in different processes, then you will need to use the 4D **CALL PROCESS** and **Outside call** commands and interprocess variables to communicate between the two processes.

DestDataType — Integer, 1 or 2. Indicates what type of data was the destination of the last drag. Values are shown below.

1	Row
2	Column
3	Cell

Example:

```
  `eSrcALP script
C_LONGINT(vDstArea;vDstID;vDstType;vRow)

Case of
  :(ALProEvt=-5)  ` User dragged a row
    AL_GetDrgSrcRow(eSrcALP ;vRow)
    AL_GetDrgArea(eSrcALP ;vDstArea;vDstID)
    If (vDstArea=eSrcALP ) ` if dragged within the same area
      AL_GetDrgDstTyp(eSrcALP;vDstType) ` get the type of data that was
                    destination of the drag
      If(vDstTyp=1) ` if dragged into a row
        AL_GetDrgDstRow(eSrcALP;vRow) ` get the row
      End if
    Else `dragged to a different area
      ◊vDstArea:=vDstArea
      CALL PROCESS(vDstID)
    End if
End case


  `Destination ALP layout’s layout proc
C_LONGINT(vRow;vDstType)

Case of
  :(Outside call)  ` Outside call (via CALL PROCESS)
    If(◊vDstArea=eDstALP) ` has a drag occurred from another process into
                    this AreaList Pro object
      AL_GetDrgDstTyp(eDstALP;vDstType) ` get the type of data that was
                    destination of the drag
      If(vDstTyp=1) ` if dragged into a row
        AL_GetDrgDstRow(eDstALP;vRow) ` get row
      End if
    End if
End case
```

AL_GetDrgDstRow

AL_GetDrgDstRow*(AreaName;DestRow)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
DestRow	integer	row number in area that was dragged to

If the destination of the last drag was a row or a cell (See "***AL_GetDrgDstTyp***" on page 140), use this command to determine which row or cell was the destination of the last drag. The *AreaName* parameter should be the destination (receiver) area of a drag.

If the destination and source areas are actually the same area or different areas within the same process (i.e., they reside on the same layout), this command may be called from the source area's script. If the destination and source areas are in different processes, then you will need to use the 4D **CALL PROCESS** and **Outside call** commands and interprocess variables to communicate between the two processes.

DestRow — Integer. This parameter returns the row number of the destination area which received the drag.

Example:

```
  `eSrcALP script
C_LONGINT(vDstArea;vDstID;vDstType;vRow)

Case of
  :(ALProEvt=-5)  ` User dragged a row
    AL_GetDrgSrcRow(eSrcALP ;vRow)
    AL_GetDrgArea(eSrcALP ;vDstArea;vDstID)
    If (vDstArea=eSrcALP ) ` if dragged within the same area
      AL_GetDrgDstTyp(eSrcALP;vDstType) ` get the type of data that was
                    destination of the drag
      If(vDstTyp=1) ` if dragged into a row
        AL_GetDrgDstRow(eSrcALP;vRow) ` get the row number
      End if
    Else `dragged to a different area
      ◊vDstArea:=vDstArea
      CALL PROCESS(vDstID)
    End if
End case


  `Destination ALP layout's layout proc
C_LONGINT(vRow;vDstType)
```

```
Case of
  :(Outside call)  ` Outside call (via CALL PROCESS)
    If(◊vDstArea=eDstALP) ` has a drag occurred from another process into
                    this AreaList Pro object
      AL_GetDrgDstTyp(eDstALP;vDstType) ` get the type of data that was
                    destination of the drag
      If(vDstTyp=1) ` if dragged into a row
        AL_GetDrgDstRow(eDstALP;vRow) ` get the row number
      End if
    End if
End case
```

AL_GetDrgDstCol

AL_GetDrgDstCol*(AreaName;DestCol)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
DestCol	integer	column number in area that was dragged to

If the destination of the last drag was a column or a cell (See "***AL_GetDrgDstTyp***" on page 140), use this command to determine which column or cell was the destination of the last drag. The *AreaName* parameter should be the destination (receiver) area of a drag.

If the destination and source areas are actually the same area or different areas within the same process (i.e., they reside on the same layout), this command may be called from the source area's script. If the destination and source areas are in different processes, then you will need to use the 4D **CALL PROCESS** and **Outside call** commands and interprocess variables to communicate between the two processes.

DestCol — Integer. This parameter returns the column number of the destination area which received the drag.

Example:

```
  `eSrcALP script
C_LONGINT(vDstArea;vDstID;vDstType;vCol)

Case of
  :(ALProEvt=-7)  ` User dragged a column
    AL_GetDrgSrcCol(eSrcALP ;vCol)
    AL_GetDrgArea(eSrcALP ;vDstArea;vDstID)
    If (vDstArea=eSrcALP ) ` if dragged within the same area
      AL_GetDrgDstTyp(eSrcALP;vDstType) ` get the type of data that was
                    destination of the drag
      If(vDstTyp=2) ` if dragged into a column
        AL_GetDrgDstCol(eSrcALP;vCol) ` get the column number
      End if
    Else `dragged to a different area
      ◊vDstArea:=vDstArea
      CALL PROCESS(vDstID)
    End if
End case


  `Destination ALP layout’s layout proc
C_LONGINT(vCol;vDstType)

Case of
  :(Outside call)  ` Outside call (via CALL PROCESS)
    If(◊vDstArea=eDstALP) ` has a drag occurred from another process into
              this AreaList Pro object
      AL_GetDrgDstTyp(eDstALP;vDstType) ` get the type of data that was
              destination of the drag
      If(vDstTyp=2) ` if dragged into a column
        AL_GetDrgDstCol(eDstALP;vCol) ` get the column number
      End if
    End if
End case
```

User Action Commands

User interaction with an AreaList Pro object is handled in the **During** phase. To accomplish this, you will most often use the various AreaList Pro commands from within an AreaList Pro object's script, which will also contain the procedures to respond to user actions such as single clicks and double clicks.

AreaList Pro's PostKey

To cause the script to execute in the **During** phase (in response to user activity), AreaList Pro has to send a message to 4D to tell it to execute the script. This is accomplished by posting a keyboard event to the Macintosh Event Queue.

The default keyboard event is Command-\. You can modify this key with the *PostKey* parameter of ***AL_SetMiscOpts*** (page 66).

Note: Versions of AreaList Pro previous to v6.0 needed an invisible button on the layout with a command key equivalent to cause the script to execute. This invisible button is no longer necessary. The invisible button may still be left in place. It will have no adverse effect on the operation of AreaList Pro. It will waste processing time and be redundant however, so it is recommended that the invisible button be removed.

Determining the User's Action on an AreaList Pro Object

AreaList Pro maintains a global variable *ALProEvt* which can be used to determine what type of user action has triggered the execution of the AreaList Pro object's script. The possible values of *ALProEvt* are:

Table 5: ALProEvt Values

Value	User Action
2	Double-Click
1	Single-Click
0	No Action
-1	Sort Button
-2	Edit Menu Select All

Table 5: ALProEvt Values

Value	User Action
-3	Column Resized
-4	Column Lock Changed
-5	Row Dragged
-6	Sort Editor
-7	Column Dragged
-8	Cell Dragged
-9	Object and Window Resized

Typically, you will use the **If**…**End if** or **Case of**…**End case** commands to check the value of *ALProEvt*. For example, if you had configured an AreaList Pro object to respond to both single and double-clicks, you might use a procedure like this in the object's script:

```
Case of
  :(Before)
    `do the setup of the AreaList Pro object here
  :(During)
     Case of
       :(ALProEvt=2)  `double-click
       :(ALProEvt=1)  `single-click
       :(ALProEvt=-1)  `sort button
       :(ALProEvt=-2)  `Edit menu Select All
       :(ALProEvt=-3)  `Column resized
       :(ALProEvt=-4)  `Column lock changed
       :(ALProEvt=-5)  `Line has been dragged from this area
       :(ALProEvt=-6)  `User has invoked AreaList Pro Sort Editor
       :(ALProEvt=-7)  `Column has been  dragged from this area
       :(ALProEvt=-8)  `Cell has been  dragged from this area
       :(ALProEvt=-9)  `Object/WIndow has been resized
    End case
End case
```

Usually you will want no processing to be performed in the AreaList Pro object's **During** phase when *ALProEvt* = 0. Since this is an external object, more **During** phases will occur than is usual for a layout object. Only those in which *ALProEvt* is non-zero have meaning for AreaList Pro processing.

If a single click is reported by AreaList Pro (*ALProEvt* = 1), and the area is in single line mode, you can determine whether the event was caused by a mouse click or by a keyboard event (the Arrow key or type-ahead scrolling). ***AL_GetColumn*** (page 150) will return zero if the event was due to an Arrow key or type-ahead scrolling.

A user double click will not cause an *ALProEvt* event if the AreaList Pro object is configured to be enterable, and the selected data entry method is via a double click. If some of the columns are not enterable, a double click on them will result in a single click *ALProEvt* event. *Please read the section “Initiating Data Entry” on page 105 for more information.*

Selection

You can determine what row or rows are selected using ***AL_GetLine*** (page 153) if in single-line selection mode, and ***AL_GetSelect*** (page 150) if in multiple-line selection mode. If you are in cell selection mode, you can use ***AL_GetCellSel*** (page 151) to determine the selected cells.

Sort Order

The user can change the sort order using the sort button (the column headers) or the sort editor. You can determine this sort order using ***AL_GetSort*** (page 149).

Column Widths

The user is able to resize the columns by clicking and dragging the dividing lines between columns. You can use ***AL_GetWidths*** (page 148) to get the width of each column, in pixels.

Column Information

AL_GetColumn (page 150) is available to determine the column where a click occured when selecting a row.

AreaList Pro allows one or more columns to be “locked” for horizontal scrolling. If the *AllowColumnLock* parameter of ***AL_SetColOpts*** (page 62) is set, the user can change the column locked (see “Column Locking” on page 12). You can determine the position of the column lock using ***AL_GetColLock*** (page 153).

Commands

AL_GetWidths

AL_GetWidths*(AreaName;ColumnNum;NumWidths;Width1; … ; WidthN)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
ColumnNum	integer	column at which to get the first width
NumWidths	integer	number of widths to get (up to 15)
Width	integer	pixel width of column

AL_GetWidths is used to get the widths of the columns to allow any user changes to the column widths to be saved for future use. Up to fifteen widths can be retrieved at a time. Global variables must be used for the *Width* parameters. Use ***AL_SetWidths*** (page 51) to override the automatic column width sizing and set the widths of a column.

ColumnNum — Integer. This parameter specifies the first column to get the width of.

NumWidths — Integer. This parameter specifies the number of widths to get. This value should be equal to the number of variables passed for the Width parameters.

Width — Integer. These parameters return the pixel widths of the columns specified by ColumnNum and NumWidths.

Example:

```
Case of
  :(Before)
    SEARCH([Prefs];[Prefs]User=Current user)
    $Error:=AL_SetArraysNam(eNames;1;4;"a1";"a2";"a3";"a4")  `display the
                    list

                    AL_SetWidths(eNames;1;4;[Prefs]Col1;[Prefs]Col2;[Prefs]C
                    ol3;[Prefs]Col4)  `get previous widths
  :(After)
    AL_GetWidths(eNames;1;4;vColumn1;vColumn2;vColumn3;vColumn4)
                    `get the current widths
    [Prefs]Col1:=vColumn1
    [Prefs]Col2:=vColumn2
    [Prefs]Col3:=vColumn3
    [Prefs]Col4:=vColumn4
    SAVE RECORD([Prefs])  `save widths in a preferences file for future use
End case
```

AL_GetSort

AL_GetSort*(AreaName;Column1; … ;ColumnN)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
Column	integer	column that sort was performed upon

AL_GetSort is used to return the current sort order.

Column — Integer. These parameters return the column or columns that the user sorted. A *Column* greater than 0 means that the column is sorted in ascending order, while a *Column* less than 0 means that the column is sorted in descending order. If a *Column* is 0 then all subsequent columns should be ignored.

The first *Column* returned will have its header underlined by AreaList Pro.

When the user sort is bypassed by setting the *UserSort* option of ***AL_SetSortOpts*** (page 69) to 2, ***AL_GetSort*** is still used to get the column header that was clicked on.

You can set the sort order using ***AL_SetSort*** (page 86).

Examples:

```
Case of
  :(ALProEvt = -1)   `user clicked a sort button
    AL_GetSort(eNames;vSortCol)   `get the sorted column
End case

$Sorted:=AL_ShowSortEd(eNames)   `display AreaList Pro Sort Editor
If($Sorted = 1)
  AL_GetSort(eNames;vCol1;vCol2;vCol3;vCol4;vCol5)   `get the sort order
    `do something here
  AL_SetScroll(eNames;1;Abs(vCol1))   `scroll to the sorted column
End if
```

AL_GetColumn

AL_GetColumn*(AreaName) ➞ ColumnNum*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
ColumnNum	integer	column that the user clicked in

Use ***AL_GetColumn*** to find out what column the user clicked in.

ColumnNum — Integer. This parameter returnes the column that the user first clicked in (mouse-down). Thus if the user clicks in column 5 and then drags the mouse and releases it in column 8, the ColumnNum returned will be 5.

Example:

```
$Column:=AL_GetColumn(eNames)   `get the column clicked on
```

AL_GetSelect

AL_GetSelect*(AreaName;Array) ➞ ResultCode*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
Array	integer array	contains element numbers selected by the user when the Multi-Line option is enabled
ResultCode	integer	identifies result condition

AL_GetSelect is used to determine which items were selected by the user when the *MultiLine* option of ***AL_SetRowOpts*** (page 59) is enabled, and they have selected multiple lines. Each element of the array contains a line number that the user selected when the list was displayed. The array *must* be an integer array, so be sure to use the **ARRAY INTEGER** command prior to calling ***AL_GetSelect***.

ResultCode — Integer. This value is equal to one (1) if enough memory was available to resize *Array*. If enough memory was not available you should react accordingly.

You can use ***AL_SetSelect*** (page 87) to highlight lines.

Example:

```
  `Multi-line option is enabled, the list is displayed, and
  `the user selects lines 2,4,5,6,10,11,15,17,18,19,25.
  `The 4D procedure looks like this:
Case of
  :(Before)
    $Error:=AL_SetArraysNam(eList;1;3;"aLN";"aFN";"aCompany")  `display
                the list
    AL_SetRowOpts(eList;1;0;0;0;0)  `turn on Multi-line option (2nd parame-
                ter)
  :(During)
    If(ALProEvt=1)  `user single-clicked
      ARRAY INTEGER(aLines;0)  `MUST use an integer array!
      $Result:=AL_GetSelect(eList;aLines)  `get the items selected by user
      If($Result=1)
        For($i;1;Size of array(aLines))  `process each array item selected by
                user
          SEARCH([Company];[Company]Name=aCompany{aLines{$i}})
            `do something here
        End for
      Else  `insufficient RAM to get user selection array
        ALERT("Running low on memory, quit and restart!")
      End if  `$Result=1
    End if  `ALProEvt=1
End case
```

AL_GetCellSel

AL_GetCellSel*(AreaName; Cell1Col; Cell1Row; Cell2Col; Cell2Row; CellArray) → ResultCode*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
Cell1Col	integer	first cell column
Cell1Row	integer	first cell row
Cell2Col	integer	last cell column
Cell2Row	integer	last cell row
CellArray	array	discontiguous cells
ResultCode	integer	identifies result condition

AL_GetCellSel is used to get the cell selection. Use the *CellSelection* option of ***AL_SetCellOpts*** (page 65) to specify a cell selection mode prior to using this command. You can procedurally set the selected cells using ***AL_SetCellSel*** (page 84).

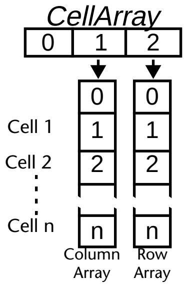
If only one cell is selected, then [*Cell1Col*, *Cell1Row*] will contain this cell and [*Cell2Col*, *Cell2Row*] will both be 0.

If more than one cell is selected and all are contiguous, then [*Cell1Col*, *Cell1Row*] and [*Cell2Col*, *Cell2Row*] will contain the starting and ending points of this range.

If more than one cell is selected but all are not contiguous, then

[*Cell1Col*, *Cell1Row*] and [*Cell2Col*, *Cell2Row*] will all be 0 and *CellArray* will contain the selected cells.

CellArray — two-dimensional integer array. The first dimension must be two. The second dimension will be set by AreaList Pro to be the same as the number of cells that are selected.



ResultCode is equal to 1 if enough memory was available to resize *CellArray*. If enough memory was not available you should react accordingly.

Example:

```
ARRAY INTEGER(aInt;2;0)
AL_GetCellSel(eList;vCol1;vRow1;vCol2;vRow2;aInt)
```

AL_GetScroll

AL_GetScroll*(AreaName;VertScroll;HorizScroll)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
VertScroll	integer	vertical position list is scrolled to (element #)
HorizScroll	integer	horizontal position list is scrolled to (column #)

AL_GetScroll returns the current position of the thumb in the vertical and horizontal scroll bars.

VertScroll — Integer. This parameter represents the element number visible at the top of the AreaList Pro display.

HorizScroll — Integer. This parameter represents the column number visible at the left of the AreaList Pro display. Both *VertScroll* and *HorizScroll* must be global variables.

The value returned in *HorizScroll* represents the actual column number, including any columns which might be currently locked. For example, if the two left columns are locked, and the list is scrolled one column to the left, so that the fourth column is adjacent to the 2nd locked column, then the value returned is four.

You can set the scroll position using ***AL_SetScroll*** (page 88).

Example:

AL_GetScroll(eNameList;vVert;vHoriz)

AL_GetColLock

AL_GetColLock*(AreaName) ➞ Columns*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
Columns	integer	number of columns that are locked

AL_GetColLock returns the number of columns currently locked.

Columns — Integer. This parameter returns the number of columns currently locked.

You can set the lock position using ***AL_SetColLock*** (page 90).

Example:

$LockColumn:=***AL_GetColLock***(eList)

AL_GetLine

AL_GetLine*(AreaName) ➞ SelectedElement*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
SelectedElement	integer	number of currently selected element

AL_GetLine returns the number of the currently selected line in the area specified by area name. ***AL_GetLine*** should only be used with an AreaList Pro object in single-line mode. If the object is in multi-line mode, you should use ***AL_GetSelect*** (page 150).

SelectedElement — Integer. This parameter returns the number of currently selected elements.

You can set the selected line using ***AL_SetLine*** (page 87).

Example:

```
   `Modify button script
   `does a MODIFY RECORD on the record corresponding
   `to the currently selected line in the AreaList Pro object eList
   `uses an ID array (previously loaded from an ID field) to load
   `the correct record
$Line:=AL_GetLine(eList)
SEARCH([Company];[Company]ID=aID{$Line})
MODIFY RECORD([Company])
```

Utility Commands

AreaList Pro includes several commands to assist in managing the operation of an AreaList Pro area.

DropArea

AreaList Pro includes a simple external area which functions as a “drop area” for rows, columns, or items dragged from other external objects. *Please read the section “Technical Details of the Dragging Implementation” on page 130 for more information.*

The DropArea is essentially an invisible object which you will place on top of a graphic image (such as a Trash can icon). You can control what types of objects can be dragged to the DropArea using two commands. If you are using the old, pre-version 5.1 method of dragging, use ***AL_SetDropOpts*** (page 184). With version 5.1 or later, use ***AL_SetDropDst*** (page 156).

DropArea Objects on a MultiPage Layout

If you are using a DropArea on a layout with multiple pages, you must disable a DropArea which is not on the active layout page. Using the pre-version 5.1 method of dragging, use ***AL_SetDropOpts*** (page 184) with values of zero (see the ***AL_SetSortOpts*** command on page 69). Using the version 5.1 dragging, use ***AL_SetDropDst*** (page 156) with null strings for the *DestCode* parameters.

Sort Editor

AreaList Pro includes a Sort Editor dialog to allow the user to sort a list using more than one column as the sort criteria. The user can command-click on the headers to display the dialog. You can use ***AL_ShowSortEd*** (page 157) to display this dialog procedurally.

Commands

%AL_DropArea

%AL_DropArea is the command used to identify the external area to which an AreaList Pro row or column can be dragged, but which does not display anything. When a row or column is dragged over this area, the area will invert.

This command will appear in the external areas popup on the 4D Object Definition Dialog when an external object is created on a layout. It is only used in the object definition for an ***%AL_DropArea*** object, and should never be used as a command in a script or procedure.

AL_SetDropDst

AL_SetDropDst*(DropAreaName;DstCode1; ... ;DstCode10)*

Parameter	Type	Description
DropAreaName	longint	name of DropArea object on layout
DstCode	string	access code(s) to be received

AL_SetDropDst is used to set the access codes for the destination of a drag, and should be called before a drag. *Please read the section “What are access “codes”?” on page 131 for more information.*

DstCode - String (15 characters). The *DstCode* can be any value (other than an empty string), such as “RowDrag”, “ColDrag”, “ALPDrag”, “PartNum”, etc. Avoid using the strings “TEXT” or “PICT”.

The code should be the same as what is passed into a potential drag partner. The drag partner will be the source/sender of the drag. The source area can be an AreaList Pro area or another external object.

The DropArea performs the following logic during the actual drag. When the drag takes place, the destination codes that were given in *DstCode1*, *DstCode2*, etc. are compared to the source codes communicated by the sender of the drag. If any of the codes match, the drag is enabled.

Please read the section “Technical Details of the Dragging Implementation” on page 130 for more information.

Note: When a DropArea is placed on a page in a multi-page layout, be sure to disable dragging for that area by calling this command with null string for the DstCode parameters. Please read the section “DropArea Objects on a MultiPage Layout” on page 155 for more information.

Example:

```
  `enable dragging a row to this area
vStr:=String(eDrop)    ` creates a unique code that only allows dragging
                  within this area
AL_SetDropDst(eList;1;vStr)    ` row type for destination
```

AL_ShowSortEd

AL_ShowSortEd*(AreaName)* → *SortDone*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
SortDone	integer	user clicked the Sort button

AL_ShowSortEd will display the AreaList Pro Sort Editor. The prompt may be set with the *SortEditorPrompt* parameter of ***AL_SetSortOpts*** (page 69). The Editor will display the header values currently specified for the AreaList Pro object. The headers for picture columns will appear, but will be disabled.

Use ***AL_GetSort*** (page 149) to determine what columns the user sorted on.

SortDone — Integer, 1, 0 or -1. This parameter returns what action the user made after the Sort Editor was displayed.

1	the user clicked the Sort button and the list was sorted
0	the user clicked the Cancel button and the list was not sorted
-1	the Sort Editor is being displayed in another process in that copy of 4D on that Macintosh. In this case you can loop until a different value is returned or continue without sorting.

Example:

```
$Sorted:=AL_ShowSortEd(eNames)   `display AreaList Pro Sort Editor
If($Sorted = 1)
  `do something here
End if
```

Changes from v4 to v5

This chapter discusses the changes from AreaList Pro v4 to v5.

Array Setup

Setting Columns in AreaList Pro

Arrays are passed to AreaList Pro using ***AL_SetArrays***, ***AL_InsertArrays***, ***AL_SetArraysNam*** (page 41), or ***AL_InsArrayNam*** (page 43). The commands are very similar, and function in essentially the same way. The new ***AL_SetArraysNam*** and ***AL_InsArrayNam*** commands, which passes the arrays by name rather than by reference, is preferred, due to an improved method of accessing the arrays. ***AL_SetArrays*** *and* ***AL_InsertArrays*** *are available for compatibility purposes with pre-version 5 procedures, and should not be used when writing new procedures.*

For both ***AL_SetArraysNam*** and ***AL_InsArraysNam***, the name of an array is passed. Except for this difference, ***AL_SetArraysNam*** functions the same as ***AL_SetArrays*** and ***AL_InsArrayNam*** functions the same as ***AL_InsertArrays***.

These new commands should be used instead of the old commands whenever possible; however, the old commands will still work fine in most cases. The exceptions to this are as follows:

- The new commands must be using when passing real arrays if the database will ever be used with the Power Macintosh native version(s) of 4D.
- The new commands must be used if the 4D command **COPY ARRAY** is used or if SQL externals are used. If the new commands are not used, then whenever the **COPY ARRAY** command or an SQL external command is used to modify arrays being displayed in an AreaList Pro object, those arrays will have to be removed using ***AL_RemoveArrays*** (page 45) and reset using ***AL_SetArrays***. If the new commands are used and all arrays are passed to AreaList Pro by name, then only ***AL_UpdateArrays*** (page 46) will need to be issued.
- ***AL_SetArraysNam*** *must* be used when passing a real array if the database will ever be used with the Power Macintosh native versions of 4D. You *must* also use this command if the 4D **COPY ARRAY** command is used with the arrays you are passing to AreaList Pro, or if any SQL externals are being

used to load the arrays.

The ***AL_SetArrays*** and ***AL_SetArraysNam*** commands can be used together. This is useful for procedures written prior to version 5 which use the ***AL_SetArrays*** command, and pass real arrays to AreaList Pro. You can simply add the new ***AL_SetArraysNam*** command, following an existing ***AL_SetArrays*** command, for real arrays only, thus minimizing the amount of rewrite required for compatibility with the PowerMac native versions of 4D.

For example, the following procedure:

```
$Error:=AL_SetArrays(eList;1;3;aArrayStr;aArrayReal;aArrayInt)
```

could be rewritten as:

```
$Error:=AL_SetArrays(eList;1;3;aArrayStr;aArrayReal;aArrayInt)
$Error:=AL_SetArraysNam(eList;2;1;"aArrayReal")
```

An alternate and preferred rewrite would be as follows:

```
$Error:=AL_SetArraysNam(eList;1;3;"aArrayStr";"aArrayReal";"aArrayInt")
```

In either case, a minimum amount of effort is required to use the new command and achieve compatibility with the PowerMac native versions of 4D.

The new ***AL_SetArraysNam*** command can be used with the new syntax for ***AL_UpdateArrays*** (page 46), which is also discussed below.

Inserting and Deleting Arrays

A new command for inserting arrays is available. The command, ***AL_InsArrayNam*** (page 43), passes the arrays by name, rather than reference. ***AL_InsArrayNam*** is functionally equivalent to the ***AL_InsertArrays*** command. Just as ***AL_SetArraysNam*** is preferred, so is ***AL_InsArrayNam***. *When using Real arrays and the PowerMac native versions of 4D, the* **COPY ARRAY** *command, or SQL externals, the new* ***AL_InsArrayNam*** *command must be used.*

Modifying Elements Procedurally

AL_UpdateArrays (page 46) is aware of the new method of passing arrays by name using ***AL_SetArraysNam*** (page 41) or ***AL_InsArrayNam*** (page 43), and as a result can directly determine the size of the arrays. When using the new commands, you can pass a value of -2 for the *NumElements* parameter, and

AreaList Pro will determine the array size for you. This is the preferred method of using ***AL_UpdateArrays*** when a redraw is required because of modifications made to a displayed array, or changes made to any attributes.

*Note: **AL_UpdateArrays** can only be passed a parameter of -1 when used from a callback procedure. Please read the section “Modifying Array Elements Procedurally” on page 27 for more information.*

Responding to User Actions on an AreaList Pro Object

No change.

Changing Pages

AL_SetDropOpts (page 184) should be used to disable a DropArea on the current page when moving to a different layout page.

The AreaList Pro Drop Area

An AreaList Pro DropArea can now be configured to accept dragged columns, rows, both, or neither, from any object, using ***AL_SetDropOpts*** (page 184).

Configuration

Headers

No change.

Footers

No change.

Column Widths

No change.

Column Locking

No change.

Rows with Multiple Lines of Text

No change.

Color

Individual array elements, called *cells*, can be assigned a unique foreground color and background color. This capability can be used to set negative numbers in red, provide special formatting to show the current selected or enterable cell, and design more attractive and useful lists. These attributes can be set in the **Before** phase, the **During** phase, and either of the AreaList Pro callback procedures.

You can use ***AL_SetCellColor*** (page 81) to set the color configuration for an individual cell, a range of cells, or a selection of discontiguous cells. ***AL_GetCellColor*** (page 85) is used to determine any cell-specific colors for a particular cell. ***AL_GetCellColor*** can only determine a color which has been set using the 4D palette of 256 colors, not the AreaList Pro palette.

Use the *MoveWithData* option of ***AL_SetCellOpts*** (page 65) to keep the cell-specific information with a cell when a row or column is dragged to a new location or the list is sorted.

The background color for a specific row can now be specified using ***AL_SetRowColor*** (page 77).

Styles

Individual array elements, called cells, can be assigned a unique font and style. This capability can be used to provide special formatting to show the current selected or enterable cell, and design more attractive and useful lists. These attributes can be set in the **Before** phase, the **During** phase, and either of the AreaList Pro callback procedures. See “Using Callback Procedures During Data Entry” on page 108.

You can use ***AL_SetCellStyle*** (page 79) to set the font and style configuration for an individual cell, a range of cells, or a selection of discontiguous cells. ***AL_GetCellStyle*** (page 83) is used to determine any cell-specific formats for a particular cell.

Use the *MoveWithData* option of ***AL_SetCellOpts*** (page 65) to keep the cell-specific information with a cell when a row or column is dragged to a new location or the list is sorted.

The font for a specific row can now be set using ***AL_SetRowStyle*** (page 75).

Sorting

No change.

Scrolling

You can hide either the horizontal or vertical scroll bar, or both, using ***AL_SetScroll*** (page 88). This allows you to construct a grid of cells, providing a different interface from a standard scrolling list.

When a scroll bar is hidden, the user is still able to scroll using the arrow keys or by dragging.

Selection

You can configure an AreaList Pro object for no cell selection, single cell selection only, or multiple cell selection, using ***AL_SetCellOpts*** (page 65). If you select not to allow cell selection, then the *MultiLines* option of ***AL_SetRowOpts*** (page 59) is used to determine the type of row selection.

When an AreaList Pro object is in cell selection mode, mouse clicks are used to highlight cells rather than rows. If multiple cell selection is enabled, then the user can shift-click and command-click to select multiple cells. Discontiguous (non-adjoining) selections are allowed.

AL_SetCellSel (page 84) is used to select cells procedurally, and can select a single cell, a range of cells, or a list of cells. You can determine the selected cells using ***AL_GetCellSel*** (page 151).

When an AreaList Pro object is in cell selection mode, it is always possible that no cells are selected.

When the user scrolls an AreaList Pro object that is in cell selection mode using the arrow keys or keyboard type-ahead, the list will scroll, but the cell selection will not change.

Row dragging is disabled when an AreaList Pro object is in cell selection mode.

The enterability options set with ***AL_SetEntryOpts*** (page 119) are fully supported when an AreaList Pro object is in cell selection mode.

If an AreaList Pro object is in multi-cell selection mode, the Edit menu Select All command is enabled.

Copy to Clipboard

The Edit menu Copy command is disabled when an AreaList Pro object has been set to allow cell selection using ***AL_SetCellOpts*** (page 65).

Drag and Drop

An AreaList Pro DropArea can now be configured to accept dragged columns, rows, both, or neither, from any object, using ***AL_SetDropOpts*** (page 184).

Using Picture Arrays

No change.

Saving and Restoring Configuration Information

Any configuration set by ***AL_SetCellOpts*** (page 62) will be saved using ***AL_SaveData*** (page 47) and restored using ***AL_RestoreData*** (page 49).

Enterability

AL_SetCellEnter (page 122) is used to set the enterability for a single cell, a range of cells, or a selection of discontiguous cells.

Precedence for configuration

AreaList Pro lets you specify the configuration for cells, rows, columns, and an entire list. The precedence used for determining how to configure an attribute is the following:

1 Cell
2 Row
3 Column
4 List

New Return Value for AL_GetColumn Command

ALProEvt will be set to one (1), and an AreaList Pro event will be generated, when the user presses the Arrow keys or performs type-ahead scrolling while in single-line mode. This functions as it did in AreaList v2.1. However, ***AL_GetColumn*** (page 150) will now return zero (0) if the *ALProEvt* was due to one of these activities. In version 2.1, ***GetAreaColumn*** returned the last clicked on column.

Changes from v4 to v5

What’s New in AreaList Pro v5 and v5.1

What’s New in AreaList Pro v5.0

- Fully compatible with the PowerMac native versions of 4th Dimension.

AreaList Pro v5 is required so that real arrays will work properly when using the new PowerMac native versions of 4D.

- Cell Selection.

Single cell and multiple cell selection is available, both procedurally and for the user. Commands let you set cell selection, as well as get cell selection.

- Cell Styles.

Single and multiple cell control to set the font and style. The font and style of a single cell can also be retrieved.

- Cell Colors.

Single and multiple cell control to set the foreground and background color. The foreground and background color of a single cell can also be retrieved.

- Specify enterability for individual cells.

AreaList Pro 5.0 allows you to set the enterability for single and/or multiple cells. You can even find out if a cell is enterable or non-enterable.

- Array Passing.

Arrays can now be passed by name as well as by reference. This ensures compatibility with 4D and other external packages (specifically SQL) by giving AreaList Pro direct access to the arrays.

- More Row Attributes.

Now includes font and background color.

- Either scroll bar can now be hidden independently.

The horizontal scroll bar can now be hidden even if all of the columns can not fit in the view. The vertical scroll bar can also hidden. The list can still be scrolled by dragging in the list or by using the arrow keys.

What’s New in AreaList Pro v5.1

- Dragging Now Uses the Macintosh Drag Manager

The Macintosh Drag Manager provides new capabilities and flexibility to the dragging interface provided by AreaList Pro.

- Dragging across processes.

You can setup an AreaList Pro object to allow dragging of a row or a column to an AreaList Pro object on a different window (which is in a different process).

- Dragging To and From CalendarSet Objects

You can setup an AreaList Pro object to allow dragging of a row or a column to a CalendarSet object, and to received drags from a CalendarSet object.

- Dragging Rows to Columns, and Columns to Rows.

You can setup an AreaList Pro object to allow a row to be dragged to a column, and a column to be dragged to a row.

What’s New in AreaList™ Pro v6

This chapter discusses the changes from AreaList Pro v5 to v6.

Displaying Fields

AreaList Pro can now display fields directly, which provides extremely fast performance even in a 4D Server environment. See “Field and Record Commands” on page 95 for the details.

AreaList Pro uses the new **SubselectionToArray** command in 4D to get the records for display. This command is available beginning with 4D v3.5.3. Therefore fields can not be displayed in an AreaList Pro object when used with an earlier version of 4D.

The user interface when fields are displayed will be essentially the same as with arrays. The few minor differences when fields are displayed are as follows:

Scrolling

Vertical scrolling will be slower when displaying fields since records have to be retrieved from 4D.

TypeAhead

Keyboard typeahead will be disabled when displaying fields.

Copy rows to the clipboard

Copying rows to the clipboard will not be allowed when displaying fields. The “Copy” menu item will be disabled when fields are displayed.

Sorting

- Indexed fields will be bold in the Sort Editor.
- Fields from related one files will be dimmed in the Sort Editor.
- Columns containing fields from a related one file will not be sorted when their column header is clicked upon.

Enterability

Columns containing fields from a related one file will not be enterable either by typing or by using popups.

The way the entry and exit callbacks are used is also somewhat different for fields than arrays. See “Executing a Callback Upon Entering a Cell” on page 109 and “Executing a Callback Upon Leaving a Cell” on page 110 for more information.

Multiple Row Dragging

AreaList Pro now supports dragging more than one row.

The user selects multiple rows by command-clicking or shift-clicking upon them. If the *DragRowWithOptKey* option of ***AL_SetDrgOpts*** (page 137) is set to 1, then the user can also select multiple rows by dragging. Once the row(s) are selected, the user may click (or option-click) to drag them. An outline of the row(s) will follow the pointer (cursor) location until the mouse is released.

To enable multiple row dragging, the following options must all be set as follows:

- The *CellSelection* option of ***AL_SetCellOpts*** (page 65) must be set to 0 (row selection is enabled).
- The *MultiLines* option of ***AL_SetRowOpts*** (page 59) must be set to 1 (multiple row selection is enabled).
- The *MultiRowDrag* option of ***AL_SetDrgOpts*** (page 137) must be set to 1 to enable multiple row dragging.

NOTE: When dragging multiple rows, there will be no automatic updating of arrays, even if the source and the destination lists are the same.

Cell Drag and Drop

The user drags a cell by clicking upon it and dragging it. An outline of the cell will follow the pointer (cursor) location until the mouse is released.

When enabled, the user can drop an item as a row, as a column or as a cell. If the destination is a cell, an outline will be shown inside the cell that the cursor is over to indicate where the item will be dropped.

PowerMac Native

AreaList™ Pro v5.2 is now PowerMac native. You must use the Mac4DX version of AreaList Pro, which includes both 68K and PowerMac native versions.

Support for MenuSet™ Menus

MenuSet interprocess menus may now be used in place of AreaList Pro popup menus. A new parameter, *MenuSetReference*, has been added to ***AL_SetEnterable*** (page 114). This reference is obtained from the MenuSet command, ***MS_ShareIPMenu***.

Increased column capacity

The maximum number of columns that can be displayed in an AreaList Pro object has been increased from 100 to 255.

Note: The maximum number of rows is 32,750. A future version of AreaList Pro will allow the display of about 8,000,000 rows.

Invisible Button

The “Invisible Button” is no longer necessary.

AreaList Pro’s PostKey

To cause the script to execute in the **During** phase (in response to user activity), AreaList Pro has to send a message to 4D to tell it to execute the script. This is accomplished by posting a keyboard event to the Macintosh Event Queue.

The default keyboard event is Command-\. You can modify this key with the *PostKey* parameter of ***AL_SetMiscOpts*** (page 66).

Note: Versions of AreaList Pro previous to v6.0 needed an invisible button on the layout with a command key equivalent to cause the script to execute. This invisible button is no longer necessary. The invisible button may still be left in place. It will have no adverse effect on the operation of AreaList Pro. It will waste processing time and be redundant however, so it is recommended that the invisible button be removed. If AreaList Pro is used with the Container™, then the invisible button must be removed.

AreaList Pro will not work properly in the Container™ if the invisible button is present.

Enhance showing/hiding of scroll bars

Two changes have been made to the way that scroll bars are shown/hidden:

- The vertical and horizontal scroll bars may now be shown by passing -2 in the *VertScroll* and *HorizScroll* parameters of ***AL_SetScroll*** (page 88). The vertical and horizontal scroll bars may now be hidden by passing -3 in the *VertScroll* and *HorizScroll* parameters of ***AL_SetScroll***.
- The horizontal scroll bar will now only be automatically shown/hidden if it has never been shown/hidden manually by passing -1, -2 or -3 in the *HorizScroll* parameter of ***AL_SetScroll*** (page 88). In previous versions there was a conflict between automatic and manual control of the horizontal scroll bar.

The possible values to use to hide or show the scroll bars are shown in the table below. The default is that both scroll bars are shown.

Value	VertScroll	HorizScroll
>0	Vertical scroll position	Horizontal scroll position
0	Hide when changing pages	Hide when changing pages
-1	Hide if shown, Show if hidden	Hide if shown, Show if hidden
-2	Show	Show
-3	Hide	Hide

Note: AreaList Pro automatically hides the horizontal scroll bar if *AllowColumnResize* in ***AL_SetColOpts*** (page 62) is set to 0 and all of the displayed columns fit within the width of the list area. AreaList Pro automatically shows the horizontal scroll bar if *AllowColumnResize* in ***AL_SetColOpts*** is set to 1 or all of the displayed columns do not fit within the width of the list area. If the horizontal scroll bar is shown or hidden manually by passing -1, -2 or -3 in the *HorizScroll* parameter of ***AL_SetScroll*** (page 88), then this behavior will be permanently disabled for the AreaList Pro object.

The default is that both scroll bars are shown and set to position 1.

Maximum Number of Draggable Objects

The maximum number of draggable objects has been increased from 50 objects to 100 objects

Up to 100 AreaList Pro and or DropArea objects may now be active and be dragged from or accept drags from other objects.

Two-Dimension Arrays

AL_SetArraysNam (page 41) and ***AL_InsArraysNam*** (page 43) now handle two-dimensional arrays

One dimension of a two-dimensional array may be passed to these commands in the Array1; … ; ArrayN parameters. For example: “My2DArray{1}” may be passed as Array1.

Disable highlighting of selected rows

A new parameter, *DisableRowHighlight*, has been added to ***AL_SetRowOpts*** (page 59).

Wrap fields when copying to the clipboard

A new parameter, *FieldWrapper*, has been added to ***AL_SetCopyOpts*** (page 68).

FieldWrapper was actually added in AreaList Pro v5.2.1 for Windows. *FieldWrapper* will be especially useful on Windows because programs such as Excel or Works expect text to be pasted in with commas separating, and quotes wrapping the fields. *FieldWrapper* is also available on the Macintosh for compatibility.

Callback Procedures for Entering or Exiting an AreaList Pro Object

Callback procedures are now provided to let you take action when an AreaList Pro object is entered or exited. See “Using the AreaEntered and AreaExited Callback Procedures” on page 28 for a complete discussion of this new capability.

Using AreaList Pro on a Resizable Window

An AreaList Pro object and its window may be made resizable using ***AL_SetWinLimits*** (page 91). Only one resizable AreaList Pro object may be placed on a layout. Other objects (4D variables, AreaList Pro objects, etc.) may be placed to the left or above this resizable object, but no objects may be placed to the right or below this object.

The AreaList Pro object is not really resizable. It appears this way because AreaList Pro draws its scroll bars at the right and bottom edges of the window instead of the right and bottom edges of the area when this option is enabled.

Please read the section “Using AreaList Pro on a Resizable Window” on page 176 for more information.

Obsolete Commands

Several commands are obsolete in AreaList Pro version 5, but are still supported for compatibilty. *You should not use these commands for new projects.*

This chapter provides a reference for these commands. Note that some commands are still used for other purposes, and are described in other chapters.

Setting Arrays

Beginning with AreaList Pro version 5, a new method of setting arrays is provided. The old methods, ***AL_SetArrays*** (page 179) and ***AL_InsertArrays*** (page 181) are still available for compatibility purposes, but should not be used for new projects. *Please read the section "Array Setup" on page 159 for more information.*

Obsolete Dragging Commands

AreaList Pro versions prior to v5.1 supported dragging of rows and columns using a proprietary technology developed for AreaList Pro. With the wide acceptance of the Macintosh Drag Manager technology, which has been incorporated into AreaList Pro, the old commands should not be used in new projects, unless the Drag Manager will not be available to you. This is likely a very rare problem, as only System 6 users are unable to use the Drag Manager.

Please read the section "Dragging Commands" on page 129 for more information.

Row Dragging

The Drag and Drop functions for row dragging are controlled by the *DragLine* parameter of ***AL_SetRowOpts*** (page 59). This parameter is used to set whether row dragging is initiated by a click or an Option-click, and to set the possible drag destinations. AL_GetDragLine is used to get the old and new position of the row, and the AreaList Pro object or ***%AL_DropArea*** object which was the destination of the row. This command should be used in the script of the source object when *ALProEvt* = -5. *Please read the section "Determining the User's Action on an AreaList Pro Object" on page 145 for more information.*

The ability of an AreaList Pro object to accept a dragged row is determined by the *AcceptDrag* parameter of ***AL_SetRowOpts*** (page 59). If the line is dragged to another position within the same list, AreaList Pro will automatically rearrange the elements of all of the arrays. If the line is dragged out of the list to another AreaList Pro object, or to an ***%AL_DropArea*** object, it is up to you to remove and insert row(s) as necessary.

Column Dragging

The Drag and Drop functions for column dragging are controlled by the *DragColumn* parameter of ***AL_SetColOpts*** (page 62). This parameter is used to set the possible drag destinations. ***AL_GetDragCol*** command is used to get the old and new position of the column, and the AreaList Pro or ***%AL_DropArea*** object which was the destination of the column. This command should be used in the script of the source object when *ALProEvt* = -7. *Please read the section "Determining the User's Action on an AreaList Pro Object" on page 145 for more information.*

The ability of an AreaList Pro object to accept a dragged column is determined by the *AcceptDrag* parameter of the ***AL_SetColOpts*** (page 62). When a column is dropped within the same AreaList Pro area, the dragged column will be automatically moved to its new position. However, it is up to you to keep track of these moves. For instance, if column 1 is dragged to the interior of the list, the column that was column 2 is now column 1, and must be addressed as such in future AreaList Pro commands. If the column is dragged outside of the list to another AreaList Pro area, or to an ***%AL_DropArea*** object, it is up to you to remove and insert column(s) as necessary.

%AL_DropArea

AreaList Pro v4.0 and v5.0 include an external area to which an AreaList Pro row or column can be dragged. This area has no display capabilities on the layout; that is, it is invisible. This enables you to use a picture area to give the ***%AL_DropArea*** the desired appearance, for example, a trash can picture can be displayed underneath the external object. When a row or column is dragged over this area, it will invert.

Dropping a row or column over this external area will cause the same *ALProEvt* values of -5 and -7, respectively, to be generated for the source AreaList Pro area. The area need not be configured in any way; these ALProEvt values will be generated without any setup being necessary. The commands

AL_GetDragLine or ***AL_GetDragCol*** can then be used to determined that the destination area was the ***%AL_DropArea*** using the *DestAreaName* parameter.

DropArea Objects on a MultiPage Layout

If you are using a DropArea on a layout with multiple pages, you must disable a DropArea which is not on the active layout page. Using the pre-version 5.1 method of dragging, use ***AL_SetDropOpts*** (page 184) with values of zero. Using the version 5.1 dragging, use ***AL_SetDropDst*** (page 156) with null strings for the *DestCode* parameters.

Commands

AL_SetArrays

AL_SetArrays*(AreaName;ColumnNum;NumArrays;Array1; … ;ArrayN) ➞ ErrorCode*

Parameter	**Type**	**Description**
AreaName	longint	name of AreaList Pro object on layout
ColumnNum	integer	column at which to set the first array
NumArrays	integer	number of arrays to set (up to 15)
Array	array	4D array
ErrorCode	integer	error code

AL_SetArrays tells AreaList Pro what arrays to display. Up to fifteen arrays can be set at a time. Any 4D array type can be used, except pointer and two-dimensional arrays. There are three very important points to note about this command:

- This command must be called first, before any of the other commands, in both the **Before** and **During** phases.
- The columns must be added in sequential order, unless the particular column has already been added. In other words, to set 30 arrays, you must set arrays 1 through 15 prior to setting arrays 16 through 30.
- All arrays set with this command must have the same number of elements as each other and as any other arrays previously set.

AL_SetArrays may be called in the **Before** phase to initially set the arrays to be displayed. Since AreaList Pro can display up to 100 arrays, this command may have to be used more than once. However, it is not mandatory to set any arrays in the **Before** phase; in that case the area on the layout where AreaList Pro is defined will be blank.

ErrorCode — Integer. The possible values are:

Value	Error Code	Action
0	No error	n/a
1	Not an array	check to make sure all arrays are correctly typed
2	Wrong type of array	pointer and two-dimensional arrays are not allowed
3	Wrong number of rows	make sure that all arrays have the same number of elements
4	Maximum number of arrays exceeded	100 arrays is the maximum
5	Not enough memory	Increase 4D's RAM partition, or change your approach to use fewer or smaller arrays

AL_SetArrays may be called in the **During** phase to set arrays to be displayed or to replace arrays that are already displayed.

Examples:

```
  `AreaList Pro eNameList script
Case of
  :(Before)
    SELECTION TO ARRAY([Contacts]FN;aFN;[Contacts]LN;aLN;[Con-
                    tacts]City;aCity;[Contacts]State;aState)  `load the arrays
    $Error:=AL_SetArrays(eNameList;1;4;aFN;aLN;aCity;aState)  `starting at
                    column 1, set 4 arrays
  :(During)
    Case of
      :(ALProEvt=2)  `user double-clicked
      :(ALProEvt=1)  `user single-clicked
      :(ALProEvt=-1)  `user sorted
End case

  `set up the eList AreaList Pro object with 25 arrays
  `two calls must be made since only 15 arrays can be passed each time
  `the arrays are already defined
$Error:=AL_SetArrays(eList;1;15;array1;array2;array3;array4;array5;array6;a
                    rray7;array8;array9;array10;
                    array11;array12;array13;array14;array15)
$Error:=AL_SetArrays(eList;16;10;array16;array17;array18;array19;array20;
                    array21;array22;array23;array24;array25)
```

AL_InsertArrays

AL_InsertArrays*(AreaName;ColumnNum;NumArrays;Array1; … ;ArrayN)* ➞ *ErrorCode*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
ColumnNum	integer	column at which to insert the first array
NumArrays	integer	number of arrays to insert (up to 15)
Array	array	4D array
ErrorCode	integer	error code

AL_InsertArrays functions the same as ***AL_SetArrays***, except that the arrays are inserted before *ColumnNum*.

All subsequent columns will maintain their settings. In other words, any header text, column styles, etc. will stay with their corresponding array.

ErrorCode — Integer. The possible values are:

Value	Error Code	Action
0	No error	n/a
1	Not an array	check to make sure all arrays are correctly typed
2	Wrong type of array	pointer and two-dimensional arrays are not allowed
3	Wrong number of rows	make sure that all arrays have the same number of elements
4	Maximum number of arrays exceeded	100 arrays is the maximum
5	Not enough memory	Increase 4D’s RAM partition, or change your approach to use fewer or smaller arrays

Example:

```
$Error:=AL_InsertArrays(eList;4;3;aFN;aLN;aComp)  `starting at column 4,
                  insert 3 arrays
```

AL_GetDragLine

AL_GetDragLine*(AreaName;OldLineNum;NewLineNum;DestAreaName)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
OldLineNum	integer	old position of the line
NewLineNum	integer	new position of the line
DestAreaName	longint	name of destination object

Use ***AL_GetDragLine*** to perform additional processing after the user has dragged a line. A line can be dragged within the list, as well as from one list to another. The source and the destination lists must both be on the same layout. The ability to accept a drag from another AreaList Pro object can also be controlled. See the parameters *DragLine* and *AcceptDrag* in ***AL_SetRowOpts*** (page 59).

If the line is dragged to another position within the list, AreaList Pro will automatically rearrange all of the arrays. If the line is dragged out of the list to another AreaList Pro object, it is up to you to remove and insert row(s) as necessary. The line may also be dragged to an ***%AL_DropArea*** object. *Please read the section “Row Dragging” on page 177 for more information.*

OldLineNum — Integer. This parameter is the position of the line before it was dragged by the user.

NewLineNum ˆ Integer. This parameter is the position of the line after the user dragged it.

DestAreaName — Long Integer. This parameter is a longint that represents the destination object.

OldLineNum, *NewLineNum*, and *DestAreaName* must all be global variables.

Example:

```
Case of
  :(ALProEvt = -5)   `user dragged a line
    AL_GetDragLine(eNames;vOldLine;vNewLine;vDestList)
    If(vDestList=eNames)
      `line was dragged within the list
    else
      `line was dragged to another object
    end if
End case
```

AL_GetDragCol

AL_GetDragCol*(AreaName;OldColumnNum;NewColumnNum;DestAreaName)*

Parameter	Type	Description
AreaName	longint	name of AreaList Pro object on layout
OldColumnNum	integer	old position of the column
NewColumnNum	integer	new position of the column
DestAreaName	longint	name of destination AreaList Pro object or ***%AL_DropArea*** object

Use ***AL_GetDragCol*** to perform additional processing after the user has dragged a column. A column can be dragged within the list, as well as from one list to another. The source and the destination lists must both be on the same layout. The column may also be dragged to an ***%AL_DropArea***. *Please read the section "Column Dragging" on page 178 for more information.*

The ability to accept a drag from another AreaList Pro object can be controlled. See the ***AL_SetColOpts*** command (page 62), and the "Column Dragging" on page 178.

When a column is dropped within the same AreaList Pro area, the dragged column will be automatically moved to its new position. However, it is up to you to keep track of these moves. For instance, if column 1 is dragged to the interior of the list, the column that was column 2 is now column 1, and must be addressed as such in future AreaList Pro commands. If the column is dragged outside of the list to another AreaList Pro area, or to an ***%AL_DropArea*** object, it is up to theyou to remove and insert column(s) as necessary.

OldColumnNum — Integer. This parameter is the position of the column before it was dragged by the user.

NewColumnNum — Integer. This parameter is the position of the column after the user dragged it.

DestAreaName — Long Integer. This parameter is a longint that represents the destination AreaList Pro object.

OldColumnNum , *NewColumnNum* , and *DestAreaName* must all be global variables.

Example:

```
Case of
  :(ALProEvt = -7)   `user dragged a column
```

```
    AL_GetDragCol(eNames;vOldCol;vNewCol;vDestList)
    If(vDestList=eNames)
      `column was dragged within the list
    else
      `column was dragged to another object
    end if
End case
```

AL_SetDropOpts

AL_SetDropOpts*(DropAreaName;AcceptRowDrag;AcceptColumnDrag)*

Parameter	Type	Description
DropAreaName	longint	name of DropArea object on layout
AcceptRowDrag	integer	accept row dragged from an AreaList Pro object
AcceptColumnDrag	integer	accept column dragged from an AreaList Pro object

AL_SetDropOpts is used to control several options pertaining to an AreaList Pro DropArea.

AcceptRowDrag — Integer, 1 or 0.

1	this DropArea object will accept a row dragged from an AreaList Pro object
0	this DropArea object will not accept a row (default)

AcceptColumnDrag— Integer, 1 or 0.

1	this DropArea object will accept a column dragged from an AreaList Pro object
0	this DropArea object will not accept a column (default)

If ***AL_SetDropOpts*** is not called, then the DropArea object will be configured to accept both rows and columns dragged from an AreaList Pro object.

You should pass a value of zero for both *AcceptRowDrag* and *AcceptColumnDrag* when the layout page containing the DropArea object is not the current layout page. *Please read the section “DropArea Objects on a MultiPage Layout” on page 179 for more information.*

Example:

```
  `setup the AreaList Pro DropArea object to accept row drags only
AL_SetDropOpts(eList;1;0)
```

Examples

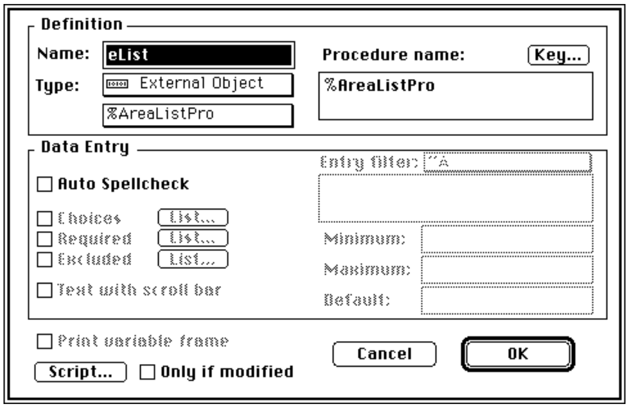
The examples in this section are designed to provide an overview of the use of AreaList Pro and the basic commands. The examples (except for #9) are included on your AreaList Pro disk, to allow experimentation with the commands and parameters.

You may also wish to examine the non-compiled version of the AreaList Pro demo, for more examples on the various AreaList Pro capabilities.

Example 1 — A Simple One-Column List

Create a simple one-column list on a 4D layout, containing the elements of a 4D List. When the user clicks on an element in the list, put the value of the selected element into a variable called *vItem*.

First we need to create the layout and draw the AreaList Pro external object. We'll name the object *eList*.



Definition

Name: eList

Type: External Object

%AreaListPro

Procedure name: %AreaListPro

Data Entry

□ Auto Spellcheck

□ Choices

□ Required

□ Excluded

□ Text with scroll bar

Entry filter:

Minimum:

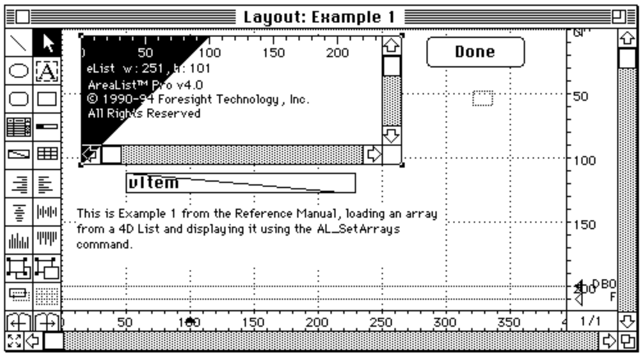
Maximum:

Default:

□ Print variable frame

□ Only if modified

Our layout now looks something like this:



Layout: Example 1

eList w: 251, h: 101

AreaList™ Pro v4.0

© 1990-94 Foresight Technology, Inc.

All Rights Reserved

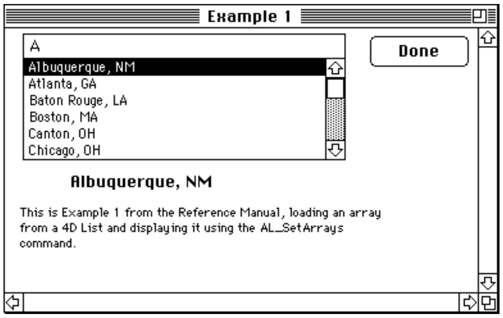
vItem

This is Example 1 from the Reference Manual, loading an array from a 4D List and displaying it using the AL_SetArrays command.

The script for the AreaList Pro object is:

```
Case of
  :(Before)  `initialize the AreaList Pro object
    LIST TO ARRAY("City, State";aCityState)  `copy the list into an array
    $errorcode:=AL_SetArraysNam(eList;1;1;"aCityState")  `display array in
                    AreaList Pro object
    vItem:=aCityState{1}  `reference the line # into array to get value
  :(During)  `respond to user action
    If(ALProEvt =1)  `did user single-click on a line?
      $Line:=AL_GetLine(eList)  `get the line the user selected
      vItem:=aCityState{$Line} `get the value in that element of the array
    End if  `ALProEvt=1
End case
```

The layout will appear like this in the User or Runtime environment:



Albuquerque, NM

This is Example 1 from the Reference Manual, loading an array from a 4D List and displaying it using the AL_SetArrays command.

Notice that the column header displays the default value of “A”. In the next example, we’ll modify the display to have a more meaningful header.

Example 2 — Displaying Headers on the List

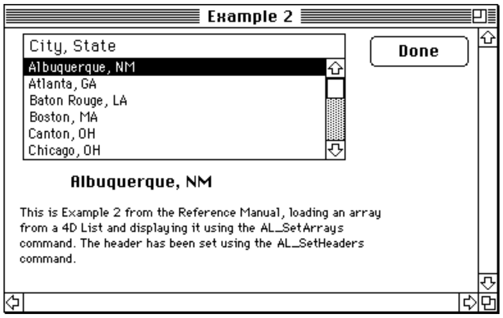
Modify the previous example to display “City, State” as the header for the list column.

The modified script for the AreaList Pro object is:

```
Case of
  :(Before)  `initialize the AreaList Pro object
    LIST TO ARRAY("City, State";aCityState)  `copy the list into an array
    $errorcode:=AL_SetArraysNam(eList;1;1;"aCityState")  `display array in
                    AreaList Pro object
    AL_SetHeaders(eList;1;1;"City, State")  `specify the value for the column
                    1 header
    vItem:=aCityState{1}  `reference the line # into array to get value
  :(During)  `respond to user action
    If(ALProEvt =1)  `did user single-click on a line?
      $Line:=AL_GetLine(eList)  `get the line the user selected
      vItem:=aCityState{$Line} `get the value in that element of the array
    End if  `ALProEvt=1
```

End case

The AreaList Pro object now appears in the User or Runtime environment like this:



City, State
Albuquerque, NM
Atlanta, GA
Baton Rouge, LA
Boston, MA
Canton, OH
Chicago, OH

Albuquerque, NM

This is Example 2 from the Reference Manual, loading an array from a 4D List and displaying it using the AL_SetArrays command. The header has been set using the AL_SetHeaders command.

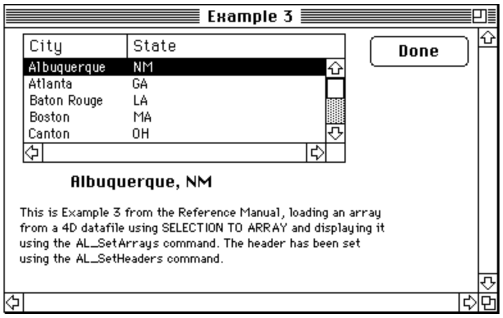
Example 3 — Displaying Data from a File

We'll change the previous example to load the array from a file in the database rather than a list. Files are commonly used to keep list items when the number of items is large or may change frequently. Also, we'll display the City and State in separate columns. This will require that our file structure keep the City and State values in two different fields. We can use the same AreaList Pro object we created in the previous examples, and just modify the script.

Our new script is:

```
Case of
  :(Before)  `initialize the AreaList Pro object
    ALL RECORDS([Cities])  `load all records in the Cities file
    SELECTION TO ARRAY([Cities]City;aCity;[Cities]State;aState)  `copy field
                values into arrays
    $errorcode:=AL_SetArraysNam(eList;1;2;"aCity";"aState")  `display
                arrays in AreaList Pro object
    AL_SetHeaders(eList;1;2;"City";"State")  `specify the values for the col-
                umn headers
    vItem:=aCity{1}+", "+aState{1}
  :(During)  `respond to user action
    If(ALProEvt =1)  `did user single-click on a line?
      $Line:=AL_GetLine(eList)  `get the line the user selected
      vItem:=aCity{$Line}+" "+aState{$Line}  `get the value in that element of
                the array
    End if  `ALProEvt=1
End case
```

Our layout now looks like this to the user:



City	State
Albuquerque	NM
Atlanta	GA
Baton Rouge	LA
Boston	MA
Canton	OH

Albuquerque, NM

This is Example 3 from the Reference Manual, loading an array from a 4D datafile using SELECTION TO ARRAY and displaying it using the AL_SetArrays command. The header has been set using the AL_SetHeaders command.

Example 4 — Selecting Multiple Lines

In the previous examples, we've used the default single-line selection mode, which allows only one line to be selected, or highlighted, at any time. AreaList Pro can be configured to allow multiple lines to be selected, and commands are available to highlight lines procedurally, as well as determine what lines have been selected by the user.

Let's modify the previous example script to work in multiple-line mode. We'll add an additional few commands to the **Before** phase part of the script. First, we need to configure the AreaList Pro object to be in multi-line mode. Second, we'll initially display the list with no lines selected.

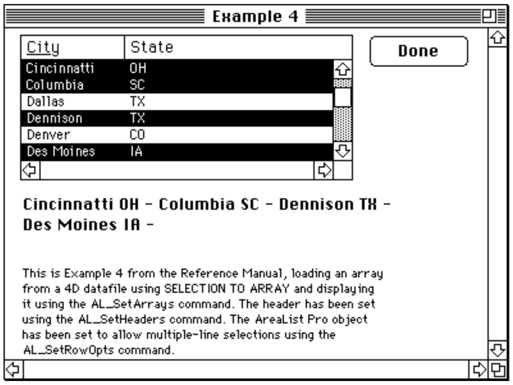
Since AreaList Pro defaults to no selected lines when in multi-line mode, we don't need to use ***AL_SetSelect*** (page 87) in the **Before** phase. When the user clicks on one or more items, we'll display the selected items in the vItem variable, separated by a dash character. Finally, if the user double-clicks on a line, we want to close the layout using the **CANCEL** command.

Here's the modified script:

```
Case of
 :(Before)  `initialize the AreaList Pro object
   ALL RECORDS([Cities])  `load all records in the Cities file
   SELECTION TO ARRAY([Cities]City;aCity;[Cities]State;aState)  `copy field
                  values into arrays
   $errorcode:=AL_SetArraysNam(eList;1;2;"aCity";"aState")  `display
                  arrays in AreaList Pro object
   AL_SetHeaders(eList;1;2;"City";"State")  `specify the values for the col-
                  umn headers
   AL_SetRowOpts(eList;1;1;0;0;0)  `Set multiline mode and allow no selec-
                  tion parameters
   vItem:=""
 :(During)  `respond to user action
   Case of
   :(ALProEvt =1)  `did user single-click on a line?
```

```
          ARRAY INTEGER(aLines;0)
          $OK:=AL_GetSelect(eList;aLines)  `get the lines selected by user
          vItem:=""
          For($i;1;Size of array(aLines))  `look at each line selected by user
           vItem:=vItem+aCity{aLines{$i}}+" "+aState{aLines{$i}}+" - "  `plug val-
                    ues in vItem
          End for
      :(ALProEvt =2)  `double-click?
         CANCEL   `cancel the layout
      End case   `ALProEvt=1
End case
```

Now our layout looks like this:



City	State
Cincinnatti	OH
Columbia	SC
Dallas	TX
Dennison	TX
Denver	CO
Des Moines	IA

Cincinnatti OH - Columbia SC - Dennison TX - Des Moines IA -

This is Example 4 from the Reference Manual, loading an array from a 4D datafile using SELECTION TO ARRAY and displaying it using the AL_SetArrays command. The header has been set using the AL_SetHeaders command. The AreaList Pro object has been set to allow multiple-line selections using the AL_SetRowOpts command.

Example 5 — Allowing Data Entry

Now that we have a basic AreaList Pro area displayed on our layout, we can implement data entry.

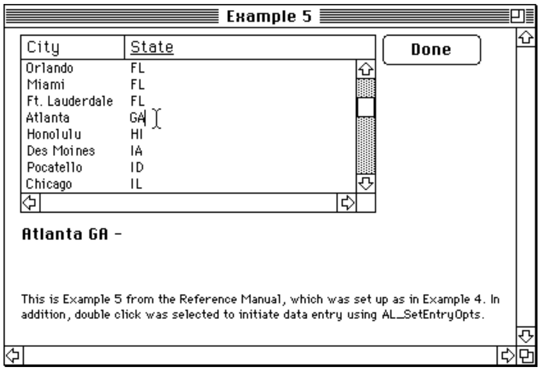
In AreaList Pro, all that needs to be done is to add a line of code to the **Before** phase of the object's script. To initiate data entry with a double click, we use ***AL_SetEntryOpts*** (page 119) with the *EntryMode* parameter set to 3 (see "Initiating Data Entry" on page 105 for more information about the different options available). As a default, AreaList Pro allows all columns to be enterable once the method of initiating data entry has been set.

The code in the script is:

```
Case of
  :(Before)  `initialize the AreaList Pro object
     ALL RECORDS([Cities])  `load all records in the Cities file
     SELECTION TO ARRAY([Cities]City;aCity;[Cities]State;aState)  `copy
                    field values into arrays
     $errorcode:=AL_SetArraysNam(eList;1;2;"aCity";"aState")  `display
                    arrays in AreaList Pro object
     AL_SetHeaders(eList;1;2;"City";"State")  `specify the values for the col-
                    umn headers
     AL_SetRowOpts(eList;1;1;0;0;0)  `Set multiline mode and allow no selec-
```

```
                tion parameters
    AL_SetEntryOpts(eList;3;0;0)  `Set double click to enter data entry mode
    vItem:=""
  :(During)  `respond to user action
    If(ALProEvt=1)  `did user single-click on a line?
      ARRAY INTEGER(aLines;0)
      $OK:=AL_GetSelect(eList;aLines)  `get the lines selected by user
      vItem:=""
      For($i;1;Size of array(aLines))  `look at each line selected by user
        vItem:=vItem+aCity{aLines{$i}}+" "+aState{aLines{$i}}+" - "  `plug val-
                ues in vItem
      End for
    End if   `ALProEvt=1
End case
```

The Layout now looks like this after double clicking on the cell with “GA” in it:



Example 5

City	State
Orlando	FL
Miami	FL
Ft. Lauderdale	FL
Atlanta	GA
Honolulu	HI
Des Moines	IA
Pocatello	ID
Chicago	IL

Atlanta GA -

This is Example 5 from the Reference Manual, which was set up as in Example 4. In addition, double click was selected to initiate data entry using AL_SetEntryOpts.

Example 6 — Restricting Data Entry to a Column

Now that data entry has been established in our example AreaList Pro area, let’s prohibit entry to one of the columns. This requires executing the ***AL_SetEnterable*** command (page 114) to override the default enterability for column 1. In this command, which is also placed in the **Before** phase, we must specify the *ColumnNumber*, which is 1, and the *Enterablility*, which we’ll set to 0 (not enterable.)

The modified script is:

```
Case of
  :(Before)  `initialize the AreaList Pro object
    ALL RECORDS([Cities])  `load all records in the Cities file
    SELECTION TO ARRAY([Cities]City;aCity;[Cities]State;aState)  `copy
                field values into arrays
    $errorcode:=AL_SetArraysNam(eList;1;2;"aCity";"aState")  `display
                arrays in AreaList Pro object
    AL_SetHeaders(eList;1;2;"City";"State")  `specify the values for the col-
                umn headers
    AL_SetRowOpts(eList;1;1;0;0;0)  `Set multiline mode and allow no selec-
```

```
                tion parameters
    AL_SetEntryOpts(eList;3;0;0)  `Set double click to enter data entry mode
    AL_SetEnterable(eList;1;0)  `Set column 1 to be non-enterable
    vItem:=""
  :(During)  `respond to user action
    If(ALProEvt=1)  `did user single-click on a line?
      ARRAY INTEGER(aLines;0)
      $OK:=AL_GetSelect(eList;aLines)  `get the lines selected by user
      vItem:=""
      For($i;1;Size of array(aLines))  `look at each line selected by user
        vItem:=vItem+aCity{aLines{$i}}+" "+aState{aLines{$i}}+" - "  `plug val-
                ues in vItem
      End for
    End if   `ALProEvt=1
End case
```

This layout looks identical to that in Example 5, except that column 1 is no longer enterable. Test this by double clicking on column 1 — the row will be selected, but you won't begin data entry. Double clicking on column two will initiate data entry as in Example 5.

Example 7 — Validating Data Entry

AreaList Pro has the capability to execute a 4th Dimension procedure when data entry ends on a cell. This is known as a callback procedure, and can be specified using ***AL_SetCallbacks*** (page 117). In this example, we add a callback procedure which checks the value entered in a column 2 cell, and warns the user if it is invalid. To implement this, ***AL_SetCallbacks*** (page 117) is called from the Before phase, and sets up the callback procedure *ExitCallback*.

The new script is:

```
Case of
  :(Before)  `initialize the AreaList Pro object
    ALL RECORDS([Cities])  `load all records in the Cities file
    SELECTION TO ARRAY([Cities]City;aCity;[Cities]State;aState)  `copy
                field values into arrays
    $errorcode:=AL_SetArraysNam(eList;1;2;"aCity";"aState")  `display
                arrays in AreaList Pro object
    AL_SetHeaders(eList;1;2;"City";"State")  `specify the values for the col-
                umn headers
    AL_SetRowOpts(eList;1;1;0;0;0)  `Set multiline mode and allow no selec-
                tion parameters
    AL_SetEntryOpts(eList;3;0;0)  `Set double click to enter data entry mode
    AL_SetEnterable(eList;1;0)  `Set column 1 to be non-enterable
    AL_SetCallbacks(eList;"";"ExitCallback")  `Set exit callback to procedure
                ExitCallback
    vItem:=""
  :(During)  `respond to user action
```

```
    If(ALProEvt=1)  `did user single-click on a line?
      ARRAY INTEGER(aLines;0)
      $OK:=AL_GetSelect(eList;aLines)  `get the lines selected by user
      vItem:=""
      For($i;1;Size of array(aLines))  `look at each line selected by user
        vItem:=vItem+aCity{aLines{$i}}+" "+aState{aLines{$i}}+" - "  `plug val-
                    ues in vItem
      End for
    End if   `ALProEvt=1
End case
```

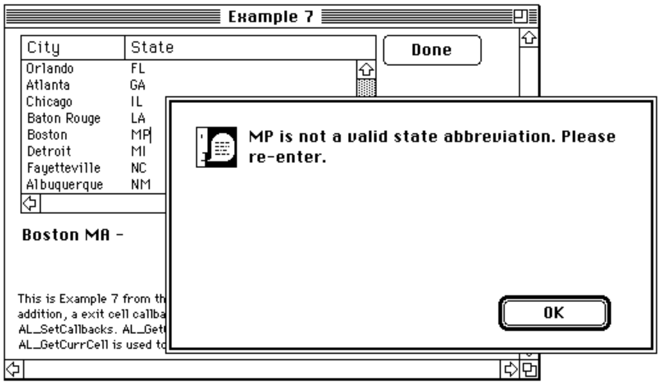
In the callback procedure, we must find out from AreaList Pro if the cell was actually modified, and if so, which cell it was. ***AL_GetCellMod*** (page 125) returns a Boolean value indicating whether the cell was modified, and ***AL_GetCurrCell*** (page 124) returns its column and row position.

Notice that the callback procedure is actually a function. AreaList Pro expects a return value which will indicate whether or not the newly entered data is accepted.

The code for the callback procedure ExitCallback is:

```
C_LONGINT($1;$2)  `must be in every callback procedure
C_LONGINT(vColumn;vRow)
C_BOOLEAN($0)
If(AL_GetCellMod(eList)>0)  `Ask AreaList Pro if cell was modified
  AL_GetCurrCell(eList;vColumn;vRow)  `Find out which cell
    `Since we only have one enterable array, we don't need to worry about
    `the column. Create a new array of all possible States.
  LIST TO ARRAY("State Abbrev";aPossStates)
  If(Find in array(aPossStates;aState{vRow})=-1)  `Is modified element not
                    valid...
    $0:=False  `Tell AreaList Pro it is invalid. This forces the user to re-enter it.
    BEEP  `Provide user feedback
    ALERT(aState{vRow}+" is not a valid state abbreviation. Please re-enter.")
  Else
    $0:=True  `Tell AreaList Pro entry is valid
  End if
Else
  $0:=True  `Tell AreaList Pro entry is valid
End if
```

In this layout, if the user were to double click into the “State” column cell with “MA” in it, enter “MP”, and exit the cell , this Alert would be displayed:



City	State
Orlando	FL
Atlanta	GA
Chicago	IL
Baton Rouge	LA
Boston	MP
Detroit	MI
Fayetteville	NC
Albuquerque	NM

Boston MA -

This is Example 7 from th
addition, a exit cell callba
AL_SetCallbacks. AL_Get
AL_GetCurrCell is used to

MP is not a valid state abbreviation. Please re-enter.

Example 8 — Prohibiting Data Entry to a Specific Cell

This final example takes advantage of the other possible AreaList Pro callback which is executed when a cell is entered for data entry. We'll make use of this callback to prohibit changes to column 1 data (City) for the state of California (abbreviation CA). The only change necessary to the AreaList Pro script is to add the entry callback procedure name, *EntryCallback*, to the call to ***AL_SetCallbacks*** (page 117).

The script now is:

```
Case of
  :(Before)  `initialize the AreaList Pro object
    ALL RECORDS([Cities])  `load all records in the Cities file
    SELECTION TO ARRAY([Cities]City;aCity;[Cities]State;aState)  `copy
                field values into arrays
    $errorcode:=AL_SetArraysNam(eList;1;2;"aCity";"aState")  `display
                arrays in AreaList Pro object
    AL_SetHeaders(eList;1;2;"City";"State")  `specify the values for the col-
                umn headers
    AL_SetRowOpts(eList;1;1;0;0;0)  `Set multiline mode and allow no selec-
                tion parameters
    AL_SetEntryOpts(eList;3;0;0)  `Set double click to enter data entry mode
    AL_SetCallbacks(eList;"EntryCallback";"ExitCallback")  `Set callback
                procedures
    vItem:=""
  :(During)  `respond to user action
    If(ALProEvt=1)  `did user single-click on a line?
      ARRAY INTEGER(aLines;0)
      $OK:=AL_GetSelect(eList;aLines)  `get the lines selected by user
      vItem:=""
      For($i;1;Size of array(aLines))  `look at each line selected by user
        vItem:=vItem+aCity{aLines{$i}}+" "+aState{aLines{$i}}+" - "  `plug val-
                ues in vItem
      End for
    End if   `ALProEvt=1
End case
```

The callback procedure code checks the column number and row information by making use of ***AL_GetCurrCell*** (page 124). Note that this procedure now makes use of the two parameters that AreaList Pro passes to it: the AreaList Pro object id, and the method by which data entry was initiated.

```
  `Global Procedure: Entry Callback
  `$1 = Arealist Pro object - passed by AreaList Pro
  `$2 = entry cause - passed by AreaList Pro
C_LONGINT($1;$2)  `must be in every callback procedure
C_LONGINT(vCurrCol;vCurrRow)
AL_GetCurrCell($1;vCurrCol;vCurrRow)
If(vCurrCol=1)
  If(aState{vCurrRow}="CA")
    AL_SkipCell($1)
  End if
End if
```

The procedure transitions data entry to the next available cell, depending on the entry cause. To accomplish this, ***AL_GotoCell*** (page 126) is used. If the user attempts to enter a non-enterable cell by clicking into it, this procedure will use ***AL_ExitCell*** (page 128) to end data entry.

```
C_LONGINT($1;$2;$AL_Object;$EntryCause)
$AL_Object:=$1
$EntryCause:=$2
$Exit:=False
Case of
  :($EntryCause=1)  `mouse click
    $Exit:=True
  :($EntryCause=2)  `tab
    vCurrCol:=vCurrCol+1
  :($EntryCause=3)  `shift-tab
    vCurrCol:=vCurrCol-1
    If(vCurrCol=0)  `leaving left-most column?
      vCurrCol:=2
      vCurrRow:=vCurrRow-1
    End if
  :($EntryCause=4)  `return
    vCurrRow:=vCurrRow+1
    If(vCurrRow>Size of array(aCity))
      vCurrRow:=vCurrRow-1
    End if
  :($EntryCause=5)  `shift-return
    vCurrRow:=vCurrRow-1
End case
If($Exit)
  AL_ExitCell($AL_Object)  `Leave data entry
Else
  AL_GotoCell($AL_Object;vCurrCol;vCurrRow)  `Goto the next enterable
                  cell
End if
```

When it is displayed in the Runtime environment, the layout in this example will look the same as the layout in Example 7. However, the cities in the first column are now enterable, except for those which are in California.

Example 9 — Dragging from AreaList Pro to CalendarSet

This is an example of dragging a row from an AreaList Pro object in one process to a CalendarSet v2 object in another process.

This example assumes that the AreaList Pro object has already been set up for dragging a row using ***AL_SetDrgSrc*** (page 135)and that the CalendarSet v2 object has already been set up for receiving a drag using ***CS_SetDrgDst***.

First, the script of the AreaList Pro object:

```
Case of
  :(During)
    Case of
      :(ALProEvt = -5)    ` a row was dragged
        AL_GetDrgSrcRow(Self»;vSrcRow)   ` get the row that was dragged

          `get the area and process that was dragged to
        AL_GetDrgArea(Self»;vDestArea;vDestProcID)

        If(vDestProcID # 1)
            `store the destination area and text in interprocess variables
          ◊destArea := vDestArea
          ◊destText := aFirstName{vSrcRow}+" "+ aLastName{vSrcRow}

            `call the other process to update the destination area
          CALL PROCESS(vDestProcID)
        End if
    End Case
End Case
```

Next, the layout procedure of the layout containing the CalendarSet v2.0 object:

```
Case of
  :(Outside call)
    Case of
      :(◊destArea = eCalArea)
          `add an element to both the date and text arrays
        INSERT ELEMENT(aCSDate;1000)
        INSERT ELEMENT(aCSText;1000)

          `get the date of the day that was dragged to
        CS_GetDrgDstDay(eCalArea;vTempDate)
```

```
          `update the arrays with the date and the text that was set in the other
          `process
        aCSDate{Size of array(aCSDate)}:= vTempDate
        aCSText{Size of array(aCSText)}:= ◊destText

        Area_Refresh(eCalArea)   ` update the area
    End case
End case
```

Troubleshooting

This chapter lists several common problems, and their solutions, encountered when working with AreaList Pro.

When trouble-shooting a problem, use all of the tools at your disposal, including the Debug window. Many problems can be quickly resolved by stepping thru each line of code, and checking the values of variables and arrays.

Garbage characters are displayed for numeric arrays

- You must use ***AL_SetArraysNam*** (page 41), not ***AL_SetArrays*** (page 179) when displaying real arrays and running using a PowerMac-native version of 4D. See “Changes from v4 to v5” on page 159.

AreaList Pro is not being updated properly

- Using Arrays — Make sure that you call the array commands, ***AL_SetArraysNam*** (page 41), ***AL_InsArrayNam*** (page 43), ***AL_RemoveArrays*** (page 45), or ***AL_UpdateArrays*** (page 46), before any other AreaList Pro command.
- Using Fields — Make sure that you call the field commands, ***AL_SetFields*** (page 99), ***AL_InsertFields*** (page 100), ***AL_RemoveFields*** (page 101), or ***AL_UpdateFields*** (page 101), before any other AreaList Pro command.
- Make sure that you call an array command whenever you modify any of the arrays.
- Make sure that you call an array or field command whenever any of the commands in the AreaList Pro Configuration group are called.
- If you open a window over an AreaList Pro object and you need to call an array command, be sure to first call the **CLOSE WINDOW** command.
- Make sure that you do not have the same arrays displayed in two active AreaList Pro objects.
- Make sure that you do not have two active AreaList Pro objects with the same name.

AreaList Pro’s scroll bars show up on other pages

- See “Changing Layout Pages” on page 38.

AreaList Pro reports wrong drop object after Drag and Drop

- See “Changing Layout Pages” on page 38.

AreaList Pro does not respond to single or double clicks

- See “Determining the User’s Action on an AreaList Pro Object” on page 145.
- An AreaList Pro object cannot be displayed on an input layout entered via the **DISPLAY SELECTION** command.
- AreaList Pro will not generate double-click events when data entry is initiated via a double click. In this case, clicking on non-enterable columns will only generate single-click events.

AreaList Pro user event code runs more than once

- Check to make sure that the code is in the AreaList Pro object’s script and not in the layout procedure. Also insure that the code runs only when *ALProEvt* is not zero (0).

Row dragging doesn’t work.

- Check the selection mode — row dragging isn’t enabled when an AreaList Pro object is in cell selection mode.
- When attempting to open a 4D window to display a layout from a callback procedure, two windows open. This can be prevented by using a variable as a flag to identify the second time the callback procedure runs, so you can avoid opening the window a second time.

A compiler run-time error occurs with a message that the parameters are undefined.

- You must define the $1, $2, and $3 parameters in the entry started and entry finished callbacks to be long integers: See “Using Callback Procedures During Data Entry” on page 108 and “Enterability” on page 96.

Example:

C_LONGINT($1;$2$3})

You will only need to define $3 if you are using fields. If you are using arrays $3 does not need to be defined.

The compiler generates warnings that parameters

are missing from AreaList Pro commands.

- This is expected behavior, caused because the compiler checks to see the number of parameters defined for an external command. Since AreaList Pro has several commands which allow a variable number of parameters, the compiler will generate a warning. These warnings can be ignored.

The footers for an AreaList Pro object aren't being displayed.

- You must correctly set the *ShowFooters* option of the ***AL_SetMiscOpts*** (page 66) command.

An AreaList Pro popup menu doesn't display the correct values after being procedurally updated.

- ***AL_SetEnterable*** (page 114) should be called when any changes are made to a *PopupArray*.

AreaList Pro doesn't display correctly, or crashes, when used with a SEARCH BY LAYOUT *command.*

- 4D does not support external areas properly on a search by layout screen. AreaList Pro will not work in this case.

My columns appear blank when displaying fields in AreaList Pro?

- You must define a global procedure called *Compiler_ALP*. You will need to paste in the code provided by the text file, Compiler_ALP, that comes with AreaList Pro v6.0. After you have created and saved this procedure quit 4D and restart your database. See “Temporary Arrays” on page 95.

Other

- When debugging database crashes due to an external, check for external resource conflicts. De-install all externals from the Structure file, then re-install them, re-installing AreaList Pro last. If using a Proc.Ext, create a new one and re-install all externals, installing AreaList Pro last. Always use the AreaList™ Pro Installer to install AreaList Pro.

AreaList™ Pro Demo

The AreaList Pro Demo database allows the various features of AreaList Pro to be experimented with. The structure is provided in both compiled and non-compiled form. Feel free to use any of the procedures in your own code; there are several useful routines included.

The compiled version of the AreaList Pro Demo may be freely distributed. The AreaList Pro floppy disk includes a compressed version of the compiled AreaList Pro Demo, which contains a demo version of the AreaList Pro external. Please don’t distribute the source code to the AreaList Pro Demo, or the non-compiled AreaList Pro Demo.

Technical Support

Foresight Technology provides support for technical questions via electronic mail and telephone. Primary support is offered via electronic mail and our World Wide Web server.

Electronic Mail

Electronic Mail: **support@fsti.com**

World Wide Web

World Wide Web: **http://www.fsti.com/**
ftp: **ftp.fsti.com**

Our World Wide Web server has a wealth of technical information available to you, including FAQ's (Frequently Asked Questions), updates, demos, and other resources.

FAX

(817) 731-9304

Telephone

Telephone support is offered, although we strongly encourage the use of electronic mail whenever possible, because we've found that taking the time to write down the question or problem can very often provide the solution before involving technical support personnel.
(817) 731-4444

Our technical support department is available to help you solve problems in the operation and use of AreaList Pro. Please be aware that we cannot design your database, but we are happy to respond to your technical questions. And always, please refer to your manuals (especially the Troubleshooting section) as your first source of information.

About Foresight Technology

Foresight Technology, Inc. is the leading publisher of 4th Dimension development tools and utilities. We also provide high-quality, integrated, and complete systems for businesses, utilizing Macintosh, 4th Dimension, and the “C” programming language. Incorporated in August of 1988, and a Registered 4D Developer since March 1988, we have consulted with and developed custom database applications for a variety of clients, including Apple Computer, CITGO Petroleum, Linbeck Construction, Taco Bell, Pizza Hut, Electronic Data Systems, Gifford-Hill Pipe Company, and many others.

Opportunities

We are always looking for enthusiastic, hard-working individuals with experience in 4th Dimension, SQL, C, C++, Mac ToolBox, Visual Basic, or Windows API. Our work environment is relaxed, yet professional, and performance and results are highly valued.

If you are interested in discussing your qualifications and interests, please contact us.

About Foresight Technology

AreaList Pro Command Reference — by Chapter

AreaList Pro Command Reference — Alphabetical

Index

Index

Index

Index